A Service Selection Conflict Avoidance Algorithm based on Multilevel Backtracking

Huaizhou Yang
College of Computer Science
Xi'an Shiyou University
Xi'an 710065, China
yanghuaizhou@sina.com

Xuelong Wang
College of Computer Science
Xi'an Shiyou University
Xi'an 710065, China
xlwang@xsyu.edu.cn

***Abstract*—When there are a lot of candidate component services in Web services composition, the suitable component services need to be selected. The service selection conflicts often occur due to the functional dependency relationships between component services. Therefore, a service selection conflict avoidance algorithm is presented. First, a formal service selection model is presented to reflect the component services, the process of service composition and the functional dependency relationships of component services. Then, based on the presented model, a service selection algorithm to avoid service conflicts is designed by using a conflict mediation mechanism, which supports multilevel backtracking and service reselection. Finally, the performance of the algorithm is tested by simulation experiments. The experiment results indicate that a service composition without any service conflict can be accomplished rapidly via the presented algorithm.**

Keywords-Web services composition; service selection conflict; service selection algorithm; adaptive system

I. INTRODUCTION

The Web service composition system is consisted of many activities. To every activity, there could have numbers of candidate component services to realize it. How to select a suitable component service for every activity is called service selection problem. In some researches [1,2], each service selection is regarded as an independent task, i.e., the service selections do not influence each other. However, in the actual applications, there often exist some dependent relationships between the component services. For example, a special bank charge service only accept some given credit card types. Therefore, there often exist some constraints between the service selections. In order to avoid service selection conflicts and adapt to different composition structures, a service selection conflict avoidance algorithm is presented in this study. First, a service selection model is presented to accurately describe the candidate component services and their dependent relationships. Then, based on the presented model, an algorithm is designed to get a feasible composition service without any conflict.

II. RELATED WORKS

The service selection conflicts are usually caused by the dependent relationships of component services. Due to the complexity of service selection problem, service selection conflicts are often ignored or simplified in some researches. For example, based on Petri net, Xiong et al. [3] present a configuration net to describe the functional dependency relationships of component services. Unfortunately, service selection conflict is not considered in their study. In some other researches [4-6], the different kinds of algorithms, such as genetic algorithm, ant colony optimization and particle swarm optimization, are used to accomplish global service selection optimization. However, in these works, only sequence composition structure is considered, while other loop, choice and concurrent structure are ignored. Therefore, the universality of these research results is restricted.

The researches aiming at the problem of service selection conflict are still insufficient at present. Danilo et al. [7] present a mediation method to solve the conflicts of service QoS and dependent relationships in service selection process. However, service selection conflicts are not guaranteed to be eliminated completely by their mediation method. In [8], a smallest conflict algorithm is presented to get optimized global service selection result. This algorithm needs a feasible solution as input, but how to get a feasible solution is not clear. This problem is studied as emphases in this paper. In [9], a genetic algorithm with repair mechanism is presented to solve service selection conflict problem. This algorithm increases the possibility to obtain a feasible solution, but conflict mediation can only be made by service reselection to a single activity. In addition, it is not guaranteed to eliminate all conflicts. In [10], a method of sorting service selection results by the conflict degrees is used in a genetic algorithm, and the multi-objective QoS conflicts can be decreased by this algorithm. However, the conflicts produced by service dependent relationships can not be solved by this method.

Being different from the related works, in the algorithm presented in this study, the mediation space of conflicts is enlarged step by step by multilevel backtracking. Therefore, the mediation of service selection conflicts can be completed successfully, i.e., a composition service without any conflict of component service can be obtained.

III. THE PROBLEM OF SERVICE SELECTION CONFLICT

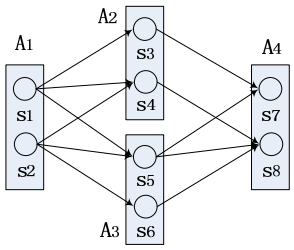

Figure 1. An example of service selection conflict .

As shown in Fig.1, a composition service is composed of four activities from A1 to A4. A2 and A3 run in concurrent mode. Each activity has two candidate component services. There are functional dependency relationships between the component services. Suppose we select component services in the order of A1 to A4, if the select result is (s1, s3, s5, s8), then there is a service selection conflict between s3 and s8 because there is not a service dependent relationship between them. In the example, the feasible selection results include (s1, s3, s5, s7), (s2, s4, s5, s8), etc. The reason to produce service selection conflicts is that there are multiple pre-activities to some composition service activities. For example, A4 has two pre-activities, which are A2 and A3. The common used loop and concurrent service composition structures will cause the situation of multiple pre-activities. In this situation, no matter which selection order is chosen, the service selection conflicts may happen. In sequence structure, because there is only one pre-activity to every activity, no service selection conflicts will happen.

The goal of service selection usually is to improve the QoS of composition service [11]. However, the functional correctness of composition service is the precondition of further QoS optimization. In addition, many algorithms for composition service optimization need a conflict-free feasible solution as initial input. Therefore, the problem of service selection conflict need to be solved firstly.

IV. Service Selection Model

According to the functions of component services, we divide them into different service classes. Every service class, as a set of candidate component services with same function, is associated with a activity of composition service. The service selection is to choose a suitable component service from each service class for every composition activity. In order to describe the functional dependency relationships between component services accurately, firstly, the service class connection are defined as follow:

Definition 1 The service class connection.

Suppose A_1 and A_2 are two composition activities, whose corresponding service classes are C_1 and C_2 respectively. If A_1 is the previous activity of A_2, then there exist a service class connection $C_1 \rightarrow C_2$ between C_1 and C_2. $C_1 \rightarrow C_2$ represents C_1 is adjacent to C_2 and C_1 is a previous service class of C_2.

Definition 2 The service selection model.

The service selection model is a tuple (S, C, R_s, R_c).

(1) S is a set of all component services. C is a set of service classes. $C=\{C_1, C_2, \cdots, C_n\}$. $S = \bigcup_{i=1}^{n} C_i$. $\forall C_i \in C, C_i \neq \varnothing$. Each service class has at least one component service. $\forall C_1, C_2 \in C$, $C_1 \cap C_2 = \varnothing$. There are no reduplicate elements in service classes.

(2) $R_s \subset S \times S$ is a set of functional dependency relationships between component services. $R_c \subset C \times C$ is a set of service class connections. If there is a service class connection $C_1 \rightarrow C_2$, then $< C_1, C_2 > \in R_c$.

(3)If $< s_1, s_2 > \in R_s$, then $\exists C_i, C_j \in C, s_1 \in C_i, s_2 \in C_j, < C_i, C_j > \in R_c$. Two component services with dependent relationship must belong to tow service classes with service class connection, respectively.

(4) $\forall C_1, C_2 \in C$, if $C_1 \rightarrow C_2$, then $\forall s_1 \in C_1$, $|\{< s_1, s > | s \in C_2\}| \geq 1$, and $\forall s_2 \in C_2, |\{< s, s_2 > | s \in C_1\}| \geq 1$. To any component service in a service class, in another service connection class, there is at least one component service which has dependent relationship with this component service.

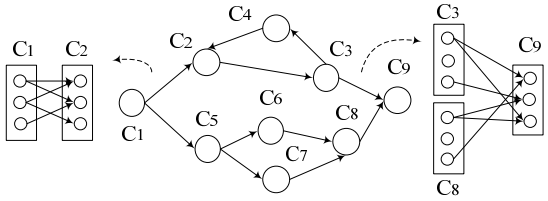

Figure 2. An illustration of service selection model .

An illustration of service selection model is shown in Fig 2. In a service selection model, (C, R_c) forms a directed graph, in which the set of vertexes is C and the set of directed edges is R_c. In addition, to each service class connection, there are a series of functional dependency relationships between corresponding candidate component services. All component services and their dependent relationships are reflected by S and R_s, respectively. According to the system activities, the process structure, the candidate component services and their dependent relationships, the service selection model of a composition service system can be constructed conveniently.

V. Algorithm Design and Analysis

Based on the service selection model, We design and analyze a service selection conflict avoidance algorithm in this section.

A. Service Class Traverse and Service Selection

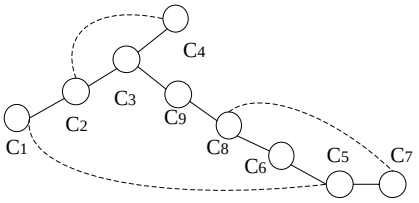

Figure 3. The service class traverse based on depth-first search .

The method of depth-first search (DFS) is used to traverse the service classes in the algorithm. DFD runs in the undirected graph constructed by (C, R_c) in the service selection model. This is because the dependent relationships of component service and the connection relationships of corresponding service classes have same directions. Only dependent relationships but directions need to be considered in the service selection process. The traversal process can start at any service class. A DFS result of corresponding undirected graph of Fig.2 is shown in Fig.3. C_1 is the startup point. The service class selection in traversal process follows

the alphabetical order of class name. the dotted lines in Fig.3 represent the untraversed service class connections.

We select the component services in the traverse order of service classes. Let C_C be the current visiting service class. C_f represents the pre-class of C_C. If a component service $s_f \in C_f$ has been selected, then the set of current candidate component services, denoted as H, can be constructed by expression (1).

$$H=\begin{cases} C_c & \text{if } C_c \text{ is initial service class} \\ \{s \mid s \in C_c \wedge (< s_f, s > \in R_s \vee < s, s_f > \in R_s)\} & \text{otherwise} \end{cases} \quad (1)$$

However, if we make service selection by the expression (1), then service conflicts may happen. This is because only one constraint coming from a selected component service in pre-class is considered, and the service dependence relationships in untraversed class connections are not considered. For example, in Fig.3, suppose $s_a \in C_2$ and $s_b \in C_4$ have been selected in the order of $(C_1, C_2, C_3, C_4, \cdots)$, if there is not dependent relationship between s_a and s_b, then a service selection conflict happens. Via analysis, we can know the service conflicts happen between the classes whose class connections are not traversed. Therefore, the set of current candidate component services should be determined by the expression (2).

$$H=\begin{cases} C_c & \text{if } C_c \text{ is initial service class} \\ \{s \mid s \in C_c \wedge (\forall s' \in M, < s, s' > \in R_s \vee < s', s > \in R_s)\} & \text{otherwise} \end{cases} \quad (2)$$

In expression (2), M is a set of component services which have been selected from the classes in connection with C_C. For example, in Fig.3, suppose the traverse order of service classes is $(C_1, C_2, C_3, C_4, \cdots)$ and $s_a \in C_2$ has been selected, when C_3 is visited, $M=\{s_a\}$. If $s_b \in C_3$ has been selected, then $M=\{s_a, s_b\}$ when C_4 is visited. The expression (2) guarantees there are not service conflicts between the current candidate component services and the selected component services. However, when $|M|>1$, due to multiple constraints, perhaps there is not any optional component service in H, i.e., $H=\varnothing$. In this situation, we present a method of multilevel backtracking and service reselection to solve this kind of problem in next subsection.

B. Multilevel Backtracking and Service Reselection

When $H=\varnothing$ in the expression (2), we need to backtrack to the parent service class of current service class and make service reselection. The service reselection must satisfy two conditions: ① The reselection process is still determined by the expression (2); ② The candidate component services to be reselected in the parent service class do not include those component services which have been selected before. The reselection process is repeated until $H \neq \varnothing$. For example, in Fig.3, when C_4 is visited, if there is not any component service in C_4, which has dependent relationships with the selected component services in C_2 and C_3, then $H=\varnothing$. In this situation, we need to backtrack to C_3 and make service reselection in C_3.

The service reselection may produce a problem. Because every service reselection process excludes all before selected component services, perhaps there is not any optional component service left in the parent service class, i.e., a service selection conflict can not be mediated in the parent service class by service reselection. In this situation, we must mediate a conflict by multilevel backtracking and service reselection. For example, in Fig.3, if a service selection conflict happens in C_4 and it can not be mediated in C_3 by service reselection, then we must backtrack to C_2 and even C_1 to mediate this conflict by service reselection. The service reselection also produces another problem. After a component service is reselected, its subsequence component services also need to be reselected. The activity to abolish a part of service selection result is equivalent to the pruning activity on the DFS graph, i.e., all the descendant nodes of service reselection node must be deleted and traversed again.

DFS, the expression (2), the multilevel backtracking, the service reselection and the pruning activity together establish a mechanism of service selection and conflict avoidance. Because the breadth-first search (BFS) will produce more pruning activities than DFS, which will influence the efficiency of algorithm, BFS is not chosen in the algorithm.

C. Prevention and Mediation of Service Selection Conflicts

The prevention of service selection conflicts is realized mainly by the expression (2). When a service class is visited in the order of service class traverse and a component service needs to be selected from this class, we use expression (2) to determine which component services are feasible for being selected. The set H in the expression (2) is a subset of the current visiting service class. In the process of constructing H, those constraints coming form the before selected component services are considered. When we select a component service from H, no any service selection conflict will happen. Therefore, when $H \neq \varnothing$, we can get a conflict-free component of a feasible solution.

The mediation of service selection conflicts is realized mainly by the mechanism of multilevel backtracking and service reselection. The conflict prevention can decrease the service conflicts, but it can not eliminate all possible conflicts. If $H=\varnothing$, then we need to change the existing selection result by service reselection. Via multilevel backtracking, we enlarge the search space step by step. Therefore, as long as there exists a feasible solution, we can find it.

In summary, through conflict prevention and mediation, a conflict-free selection result is guaranteed in each selection step. Therefore, we can get a feasible solution without any conflict in the end by the algorithm.

D. Algorithm Implementation

The main procedure of the algorithm is shown in Fig.4. Each component service selection result is recorded in the structure *ServiceSelectin*. The item *PostClass* is used to the pruning activity. The procedure *Explore( )* which is shown in Fig.5 is used to traverse the service classes and select the component services. In Fig.5, the statement labeled by "*DFS:*" calls the procedure *Explore( )* to realize DFS in a iterative mode. The statements from "*HB*" to "*HE*" are used to construct the set of current candidate component services according to the expression (2). The statements from "*SB*" to "*SE*" are used to realize the multilevel backtracking and the conflict-free service selection. The statement labeled by "*PRUNE:*" calls the procedure *Pruning( )* shown in Fig.6 to realize the pruning activity. In Fig.6, *B* is a set of service classes. The service classes in the set *B* and their descendant service classes are deleted by the procedure *Pruning( )* in a iterative mode. By executing the algorithm, we can get a service selection result without any conflict.

```
Input: A service selection model (S, C, Rc, Rs)
Output: A service selection result without any conflict
Initialization:
  Structure ServiceSelection  //define a structure
    Class
    Service
    PostClass  //a set of post service classes
  End Strucure
  Dim Stru As ServiceSelection
  Q = ∅     //a set of structure ServiceSelection
Begin
  Pick ∀Ci ∈ C
  Explore(Ci ,NULL,NULL)
  Return Q  //the set Q represents the selection result
End
```

Figure 4. The main procedure of algorithm .

E. The Complexity Analysis of the Algorithm

In the algorithm, because the main procedure selects any service class as the search startup point and calls *Explore( )* only once, the complexity of main procedure can be ignored. In the procedure *Explore( )*, before each service selection, in order to construct the constraint set M , we need to traverse the set of service class connections R_c and the set of selected component services Q once. Therefore, the complexity to construct the set M does not exceed $O(|R_c|+|C|)$. After constructing M , in order to construct the set of current candidate component services H , we need to traverse M , the set of current visiting service classes V and the set of service dependency relationships R_s . Because the average number of the component services in a service class is $(|S|/|C|)$, the complexity of this part of algorithm is $O(|M|\times(|S|/|C|)\times|R_s|)$. The expression $O(|M|\times(|S|/|C|)\times|R_s|)$ can be changed to $O(|S|\times|R_s|/|C|)$ due to $|M|$ is usually very small. According to the above analysis, before each

```
Procedure Explore( v, u, sf ) // v is current class; u is front class
          // of v in DFS; sf is the service selected form u .
Begin
HB: K = {Ci |< Ci, v >∈ Rc ∨ < v, Ci >∈ Rc} \ {u}
    K' = ∅    //a temporary set of service classes
    M = ∅     //a service constraint set of selected components
For Each k ∈ K
    For Each q ∈ Q
        If q.Class = k Then
            K' = K' ∪ {q.Class}
            M = M ∪ {q.Service}
        End If
    Next
Next
  If u =Null Then
    H = v    // a set of current candidate component services
  Else
    M = M ∪ {sf}
    H = {s | s ∈ v ∧ (∀s' ∈ M, < s', s >∈ Rs ∨ < s, s' >∈ Rs)}
HE: End If
SB: If H ≠ ∅ Then
    Stru.Service = ∀sc ∈ H
    Stru.Class = v
    Stru.PostClass = ∅
    Q = Q ∪ {Stru}
    Success =True          //Every descendant class of v has made
                           //a service selection successfully.
    For Each Ci ∈ K \ K'
      If (Ci ≠ ∀q.Class, q ∈ Q) Then
        For Each q ∈ Q
          If q.Class = v Then q.PostClass = q.PostClass ∪ {Ci}
        Next
DFS:    If Explore(                )=False Then
                Success =False
                P = {Ci | Ci ∈ q.PostClass, q ∈ Q, q.Class = v}
                Q = Q \ {Stru}
              H = H \ {Stru.Service}
                K = K ∪ P
PRUNE:          Pruning( P ) //delete all descendant classes of v
                Exit For
            End if
        End If
    Next
    If Success =True Then Return True Else GOTO SB
  Else
      Return False
SE: End If
End
```

Figure 5. The service class traverse and service selection.

```
Procedure Pruning ( B )
Begin
  For Each b ∈ B
    Pruning ({Ci | Ci ∈ q.PostClass, q ∈ Q, q.Class = b} )
    Q = Q \ {q | q ∈ Q, q.Class = b}
  Next
End
```

Figure 6. The procedure of pruning .

service selection, the complexity to construct the set H is $O(|R_c|+|C|+|S|\times|R_s|/|C|)$. The complexity of standard DFS is $O(|V|+|E|)$. $|V|$ and $|E|$, which are denoted as $|C|$ and $|R_c|$ in this algorithm, are the numbers of vertexes and edges of a graph, respectively. Each visiting to a service class will produce a service selection. Therefore, when the time to mediate the conflicts is not considered, the complexity of the algorithm is determined by expression (3).

$$O(|C|\times(|R_c|+|C|+|S|\times|R_s|/|C|)) \quad (3)$$

In the process of multilevel backtracking and service reselection, let ω be the average number of backtracking levels, then the time spend to mediate a conflict is $O((|S|/|C|)^{\omega})$. Revisiting a service class will produce a pruning activity. The procedure of pruning is a simple DFS process and its complexity is linear. In contrast with the time of service reselections, the complexity of pruning procedure can be ignored. According to the above analysis, if let the average number of service conflicts be μ, then the total time to mediate service selection conflicts is:

$$O(\mu\times(|S|/|C|)^{\omega}) \quad (4)$$

The final complexity of the algorithm is a combination of expression (3) and the expression (4):

$$O(|C|\times(|R_c|+|C|+|S|\times|R_s|/|C|)+\mu\times(|S|/|C|)^{\omega}) \quad (5)$$

VI. Simulation Experiment and Analysis

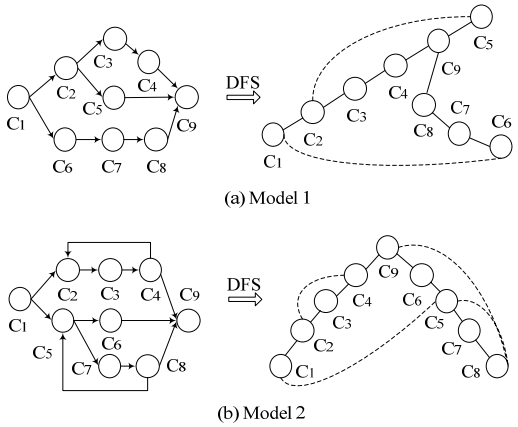

Figure 7. The experiment models and their depth-first search graphs.

Two experiment models and their DFS results are shown in Fig.7. C_1 is the startup point of DFS. The order of service class selection follows the alphabetical order of class name. Because there are more untraversed service class connections (denoted as dashed lines in Fig.7) in model 2 than those in model 1, the service selection conflicts are more likely to happen in model 2. We test the algorithm efficiency under different model complexities by model 1 and model2.

In the experiments, we produce same numbers of component services for each service class. The dependent relationships between component services are preset and divided into two types: broad dependence (BD) and limited dependence (LD). To BD, in a service class connection, each component service in pre-class is associate with n component services in post-class. The value of n is produced randomly according to $1\le n\le\xi$. ξ is the number of component services in each service class. To LD, The value of n is produced randomly according to $1\le n\le\lceil\xi/2\rceil$. In contrast with BD, LD is more likely to produce service selection conflicts due to its smaller service selection space. We test the algorithm efficiency under different numbers of service dependency relationships by BD and LD.

The experiments are made on a desktop computer with Windows 2000, 2.1 GHz CPU and 1 GB RAM. Each data is a average value of 20 experiment results.

Experiment 1: The algorithm efficiency under different model complexities and service dependence relationship numbers. The experiment result is shown in Fig.8.

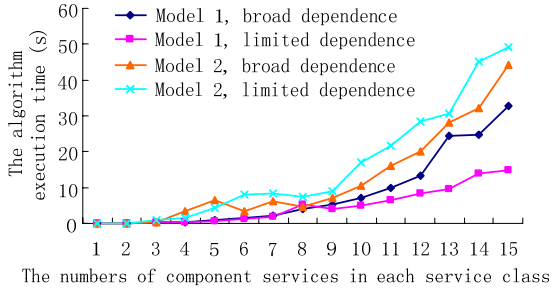

Figure 8. The algorithm execution time.

From the experiment result, we can know the algorithm efficiency is mainly determined by the numbers of candidate component services. The lower the model complexity is, the better the algorithm efficiency will be. In model 1, because the value of $|R_s|$ in LD is much smaller than that in BD, the algorithm efficiency in LD is better than that in BD. However, in model 2, the algorithm efficiency in BD is better than that in LD. This is because there are more conflicts in LD and we need to backtrack through more levels to mediate a conflict. Although the value of $|R_s|$ in LD is smaller, more time is spend to mediate the service selection conflicts in the algorithm.

Experiment 2: The influence rate of service conflict. In order to test the influence degree of conflict mediation time on the algorithm, we define η as the influence rate of service conflict. η is the ratio of conflict mediation time to algorithm execution time. The smaller the value of η is, the better the adaptability of algorithm to the conflicts will be. To focus the algorithm efficiency in the service conflict situations, we only pick the experiment instances with service conflict. The final experiment result is shown in Fig.9.

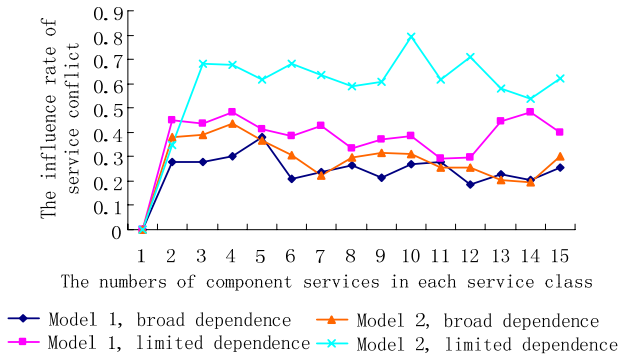

Figure 9. The influence rate of service conflict .

From the experiment result, we can know η is not affected by the numbers of candidate component services. In the four groups of results, From the experiment result, we can know η is not affected by the numbers of candidate component services. In the four groups of results, the value of η in LD is always larger than that in BD. Therefore, η is mainly determined by the numbers of service dependency relationships. In the situation of LD, the algorithm needs more time to mediate the service conflicts. To model 2, the value of η increases fast when BD changes to LD, which indicates the numbers of service dependency relationships especially affect η in the situation of complex model.

Experiment 3: The algorithm contrast. At present, the algorithms to solve service selection conflict are very scarce. In related works, MCHC (Minimal-Conflicts Hill-Climbing) is a representative algorithm [9]. Its main idea is mediating the local conflicts via an iteration procedure and eliminating the global conflicts gradually. We select it to make contrast with our algorithm. Model 1 and LD mode are selected in the experiment. The experiment result is shown in Table 1.

TABLE I. THE EXPERIMENT RESULT OF ALGORITHM CONTRAST

Service Numbers in Each Class	**Average Number of Conflicts**	**Multilevel Backtracking Algorithm**		***MCHC Algorithm***	
		Time(s)	***Mediation Rate***	***Time(s)***	**Mediation Rate**
1	0	0.02	100%	0.02	*100%*
2	0.07	0.07	100%	0.06	*99.9%*
3	1.54	0.53	100%	0.32	*87.6%*
4	2.56	0.87	100%	0.77	*81.6%*
5	3.07	1.55	100%	1.01	*63.5%*
6	3.18	2.78	100%	1.54	*52.55%*
7	4.91	3.24	100%	2.91	*46.54%*
8	5.62	4.41	100%	2.75	*37.88%*
9	6.33	4.16	100%	3.46	*24.34%*
10	7.40	5.73	100%	3.87	*26.7%*

From the experiment result, we can know the average number of conflicts increases along with the increase of candidate component services in each service class. The service selection conflicts can be mediated completely by our algorithm. However, the mediation rate of MCHC algorithm declines gradually along with the increase of conflicts. This is because the multilevel mediation method is used in our algorithm but the single-level mediation method is used in MCHC algorithm. Because the much larger local space is tried to mediate the conflicts in our algorithm, the efficiency of our algorithm is a bit lower than that of MCHC algorithm. Therefore, our algorithm is more suitable to the situation of complex service dependency relationships.

VII. CONCLUSION AND FUTURE WORK

With the enlargement of Web service composition scale and the increase of candidate component services, the fully automatic service selection and composition become an actual requirement. Because there often exist many functional dependency relationships between the component services, the service selection conflicts usually happen in the service selection process. In order to solve the problem, we present a service selection model in this study. In this model, the component services, the service classes, the service class connections and the service dependency relationships are used to reflect the service selection scenario with complex service dependency relationships. Furthermore, a conflict avoidance algorithm is presented to mediate the service selection conflicts through multilevel backtracking and service reselection. A conflict-free service composition scheme can be obtained automatically by the algorithm. In the future work, we will study how to combine our algorithm with different kinds of optimization algorithms and how to realize the global QoS optimization of service composition.

ACKNOWLEDGMENT

This research was sponsored by Shaanxi Province Research and Development Program in Science and Technology of China under the grant No. 2011k06-33.

REFERENCES

[1] B. Li, Y Xu, J Wu, and J.W. Zhu, “A Petri-net and Qos based model for automatic Web service composition,” Journal of Software. 2012, 7(1): 149-155.

[2] H.Y. Hu, W.Q. Dong, R. Fu, X. Zhang, and X.Y. Zhao, “Pareto optimality based genetic algorithm in Web services composition,” Journal of Xi'an Jiaotong University. 2009,43(12): 50-54.

[3] P.C. Xiong, Y.S. Fan, and M.C. Zhou, “QoS-aware Web service configuration,” IEEE Transactions on Systems, Man, and Cybernetics. 2008, 38(4): 888-895.

[4] Y.B. Wu and X. Wang, “Applying multi-objective genetic algorithms to Qos-aware Web service global selection,” Advances in Information Sciences and Service Sciences. 2011,3(11): 473-482.

[5] Q.Q. Fang, X.M. Peng, Q.H. Liu, and Y.H. Hu, “A global QoS optimizing Web services selection algorithm based on MOACO for dynamic Web service composition,” 2009 International Forum on Information Technology and Applications. Chengdu, China: IEEE Computer Society, 2009: 37-42.

[6] S.G. Wang, Q.B. Sun, and F.C. Yang, “Web service dynamic selection by the decomposition of global QoS constraints,” Journal of Software. 2011,22(7): 1426-1439.

[7] A. Danilo and P. Barbara, “Adaptive service composition in flexible processes,” IEEE Transactions on Software Engineering. 2007,33(6): 369-384.

[8] H.F Li, X.X Yang, and Y.Q. Ouyang, “MCHRC: min-conflict heuristic based Web services chain reconfiguration approach,” Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE2009). Wuhan, China: IEEE Computer Society, 2009: 508-511.

[9] L.F. Ai and M.L. Tang, "QoS-based Web service composition accommodating inter-service dependencies using minimal–conflict hill-climbing repair genetic algorithm," Proceedings of the Fourth IEEE International Conference on eScience. Indianapolis, USA: IEEE Computer Society, 2008: 119-126.

[10] L.F. Ai and M.L. Tang, "A penalty-based genetic algorithm for QoS-aware Web service composition with inter-service dependencies and conflicts," Proceedings of the 2008 International Conference on Computational Intelligence for Modelling Control & Automation (CIMCA'08). Vienna, Austria: IEEE Computer Society, 2008: 119-126.

[11] J.E. Haddad, M. Manouvrier, and M. Rukoz, "TQoS:transactional and QoS-aware selection algorithm for automatic Web service composition," IEEE Transactions on Services Computing. 2010,3(1): 73-85.