A Novel Web Service Composition Algorithm for Multiple QoS Constraints

Changsong Liu, Dongbo Liu, Ning Han
School of Computer and Communication, Hunan Institute of Engineering, Xiangtan, 411104, China
Email: liuchangsong74@126.com

Abstract—**Web services are creating unprecedented opportunities for the formation of online Business-to-Business collaborations. However, effective approach to service composition with multiple QoS constraints still remains an open issue, since both the QoS performance of services and the QoS requirements of applications might dynamically be changed at runtime. In this paper, we introduce a novel service composition algorithm which formulates the service composition problem as a Multiple choice Multiple dimension Knapsack Problem (MMKP) and applies evolution strategy to obtain an approximate solution. Experimental results compare our method with other solutions and demonstrate the effectiveness of our approach toward the identification of an optimal solution to the QoS constrained Web service selection problem.**

Index Terms—**web service, quality of service, service composition, workflow, QoS negotiation**

I. Introduction

Web services are autonomous components that can be advertised, located, and accessed through XML based standards and transmitted using Internet protocols [1, 4]. The emergence of Web services has created unprecedented opportunities for organizations to establish various collaborations with other organizations. For example, an integrated financial management Web service can be created by composing more specialized Web services for payroll, tax preparation, and cash management. Since, they are intended to be discovered and used by other applications across internet, Web services need to encapsulate application functionality and information resources, and make them available through programmatic interfaces [2, 18, 19]. In the presence of multiple Web services with similar functionality, users will discriminate these alternatives based on their quality of service (QoS) requirement. Therefore, Web services need to be described and understood both in terms of functional capabilities and quality of service properties [5, 8-11].

The early solutions select Web services by associating the running activity to the best candidate service which supports its execution, which can only guarantees those local QoS constraints such as the price of a single Web service. Recently, many global solutions are proposed to satisfy the process constraints and user preferences for the whole application. In this way, QoS constraints can predicate at a global level, for example, the constraints posing restrictions over the whole composed service execution can be introduced. However, those global approaches introduce an increased complexity with respect to local solutions, also, their performance tend to be variable since workload on Web services are fluctuated dramatically. For instance, when a business process has a long duration, the set of services may change their QoS properties during the process execution or some services can become unavailable. So, a dynamic and adaptive composition approach is needed, in which runtime changes in the QoS of the component services are taken into account.

In this paper, we introduce a novel service composition algorithm which formulates the service composition problem as a Multiple choice Multiple dimension Knapsack Problem and applies evolution strategy to obtain an approximate solution. The rest of this paper is organized as follows: Section 2 presents the related work; Section 3 presents the definition of problem and designs the corresponding algorithm. In Section 4 presents, extensive experiments are conducted to verify the effectiveness and performance of the proposed algorithm compared with other approaches. Finally, Section 5 concludes the paper with a brief discussion of future work.

II. Related work

The standard description of service composition are proposed in ebXML [18] and DAML-S [3]. DAML-S supports the Semantic Web services based on a generic ontology, in which both functional and QoS aspects of services are expressed as rule-based preconditions and post-conditions on service operations. However, neither DAML-S nor ebXML consider user's QoS criteria, nor do they address the issue of dynamic service selection and adaptive service composition.

The technique of Web service composition has received a lot of attention. Most of the work can be broadly classified into three categories: manual composition, semi-automatic, and automatic composition. In manual composition, the composite service is modeled manually by using a service flow language such as BPEL4WS [2] and they often requires the knowledge of specific domain. Meanwhile, it is a labor-intensive and error-prone task; so, it is not appropriate for the large-scale web service composition. In [6, 12, 19], some Semi-automatic composition techniques have been proposed, which solve some of the problems of manual composition.

The technique of semantic Web has been proven to be automatic and flexible since a composed service process is built automatically from a high level specification of the required functionality. In [13], Lazovik et al. designed a framework which interleaves planning and execution of complex applications whose functional objectives and QoS requirements are specified by assertions of XSAL languages. In [8], Dacosta et al. proposed a similar approach which built complex applications from a high level workflow specification by applying contingency planning technique. However, the drawback of these techniques is high computation complexity, which often means that only suboptimal solution can be obtained by some heuristic functions.

The technique of agent-based service composition has recently received great attention. For example, McIlraith and Son [15] proposed an agent-based web service composition framework; Maamar et al. [14] presented an agent-based and context-oriented approach that supports the composition of web services; Wang et al. [22] proposed an agent-based web service workflow model for inter-enterprise collaboration. During service composition process, software agents engage in conversations with their peers to agree on the web services that participate in this process. However, the agents in the above mentioned approaches are not used to achieve the automatic web service composition but to the coordination and enactment of the composite web services.

III. The Framework and Descriptions

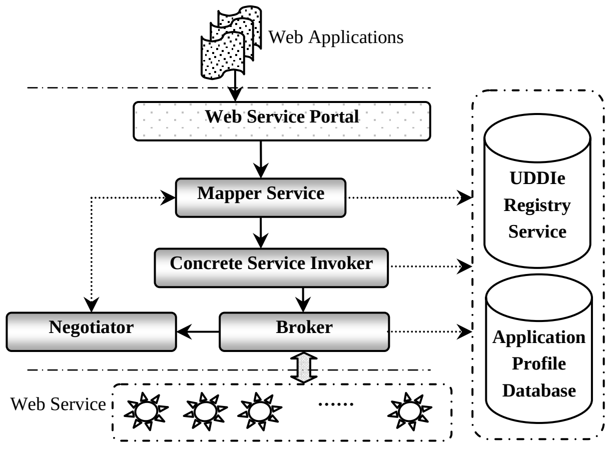

Fig. 1. Framework of QoS-aware Web Service Composition

The adaptive service composition algorithm presented in this paper is based on the framework as shown in Fig. 1. In the framework, user applications invoke Web services through *Web Service Portal*, in which the service abstract interface requirements and user's QoS constraints are specified. The registry service is an extended version of a UDDI registry, which provides related information on service abstract interface and corresponding QoS characteristics. The *Concrete Service Invoker* is responsible for invoking the concrete service corresponding to application's requirements through a *Mapper* component. Then the *Mapper* selects the best concrete services for each task of the composed service according to the optimization criteria which will be discussed in the remainder of this paper. The *Broker* component is responsible for invoking services that selected by *Mapper*, and QoS negotiation is performed through the *Negotiator* component. In order to provide adaptive QoS for user applications, we designed an application profile database which includes the context of application invocation, application performance profile and service performance profile. As the invocation context can vary over time, by this profile database, different services can be invoked due to modifications of the context.

A. Definitions on Service Execution

A Web service is modeled as a software component that implements a set of operations. In the framework, a composed service is specified at an abstract level as a high-level business process, which contains an initial task t_{init} and an exit task t_{exit}. So, an abstract composed service can be noted as a directed graph G=<$\boldsymbol{T}$, $\boldsymbol{E}$>, where $\boldsymbol{T}=\{t_1, t_2, \ldots, t_n\}$ is the set of tasks, $\boldsymbol{E}=\{e_{i,j} | \text{if } <t_i,t_j> \in \boldsymbol{T}\times\boldsymbol{T}\}$. As mentioned above, the *Mapper Service* is responsible for transforming abstract business process into concreted composed service. So, a concreted composed service can also be noted as a directed graph G^*=<$\boldsymbol{S}$, $\boldsymbol{P}$>, where $\boldsymbol{S}=\{ws_1, ws_2, \ldots, ws_n\}$ is the set of Web services, $\boldsymbol{P}=\{p_1, p_2, \ldots, p_n\}$ is the set of execution path. When a task t_i is mapped onto s_j and invokes the k-th operation interface, it is noted as a pair < t_i, $ws_{j,k}$>. Invoking path $ip_k=\{p_i,\ldots,p_j\}$ is a set of order execution path, which starts from p_i and ends at p_j.

B. QoS Measurements of Web Services

Several QoS measurements can be associated when executing a Web service. In this paper, we assume that the values of QoS measurements are real numbers that vary in a bounded range with a minimum and a maximum value. If the same operation is accessible from the same Web service and the same provider, but with different quality characteristics, then multiple copies of the same operation will be stored in the registry, each copy being characterized by its quality profile.

We mainly take into account the following subset of QoS measurements, which have been the basis for many QoS-aware service architectures [9, 10, 15, 20]: (1) $Avail_{i,j}$: The probability that the service operation $ws_{i,j}$ is accessible, which is a number in the range [0, 1]; (2) $Cost_{i,j}$: The fee that a user has to pay to the service provider for the service invocation $ws_{i,j}$; (3) $Exec_{i,j}$: The duration between the time when $ws_{i,j}$ is invoked and then time when the result is obtained; (4) $Rep_{i,j}$: It is defined as the ratio of the number of service invocations which comply the negotiated QoS to the total number of invocations, which is in the range [0, 1]. With respect to negotiability, the QoS measurement such as costs and execution time are negotiable, while availability and reputation are not negotiable. An example of QoS constraints specified by a user is shown as following.

QosConstraint.xml

```
<QosContraints name=task1>
      <CostConstraint value=100 />
      <ExecTime value=2.5 />
      <Reputation value=0.9 />
</QosConstraints>
<QosContraints name=task2>
      <CostConstraint value=70 />
      <ExecTime value=1.5 />
      <Reputation value=0.75 />
</QosConstraints>
<QosContraints name=task3>
      <CostConstraint value=110 />
      <ExecTime value=5.0 />
      <Reputation value=0.95 />
</QosConstraints>
```

According to the definitions in Section III.A, the aggregated QoS measurements of a composed service can be calculated as following.

$$Cost(ip_k) = \sum_{<t_i, ws_{j,l}> \in ip_k} Cost_{i,j} \quad (1)$$

$$Exec(ip_k) = \max\left\{ \sum_{<t_i, ws_{j,l}> \in ip_k} Exec_{i,j} \right\} \quad (2)$$

$$Avail(ip_k) = \prod_{<t_i, ws_{j,l}> \in ip_k} Avail_{i,j} \quad (3)$$

$$Rep(ip_k) = \frac{1}{|ip_k|} \sum_{<t_i, ws_{j,l}> \in ip_k} Rep_{i,j} \quad (4)$$

C. Solution of Optimal Service Composition with QoS Constraints

In this paper, we focus on Web service composition with multiple QoS constraints. Based on the aggregated QoS definition in (1)~(4), the problem can be determined by solving the following optimization programming problem.

$$\max Q = \sum_{i=1}^{k} \left(Cost(ip_k) + Avail(ip_k) + Rep(ip_k) + Exec(ip_k) \right)$$
$$s.t. \; \sum_{i\in T} \sum_{j\in S} \sum_{m\in ip_k} c_{i,j}^{m} \cdot Cost_{i,j} \le QoS_{cost}$$
$$\max_{i\in T} \{ \max_{j\in S} \{ \max_{m\in ip_k} \{ e_{i,j}^{m} \cdot Exec_{i,j} \}\}\} \le QoS_{exec} \quad (5)$$
$$\frac{1}{|ip_k|} \cdot \sum_{i\in T} \sum_{j\in S} \sum_{m\in ip_k} r_{i,j}^{m} \cdot Rep_{i,j} \le QoS_{rep}$$
$$\sum_{m\in ip_k} c_{i,j}^{m} = \sum_{m\in ip_k} e_{i,j}^{m} = \sum_{m\in ip_k} r_{i,j}^{m} = 1$$

where $c^{m}_{i,j}$, $r^{m}_{i,j}$ and $e^{m}_{i,j}$, are all boolean variants which indicate that whether < t_i, $ws_{j,m}$> is in the invoking path ip_k; QoS_{cost}, QoS_{exec} and QoS_{rep} are user's QoS constraints that specified in the QosConstraint.xml file. It is clear that the above problem is a classic 0-1 integrated programming problem, which is equivalent to the *Multiple choice Multiple dimension Knapsack Problem* (MMKP). MMKP is a well-known NP-hard problem and many heuristic approaches have been proposed to solve it approximately [5, 11, 21]. In this paper, we use evolution-computing technique to solve the above problem, and the algorithm is shown as following.

Algorithm of Optimal Service Composition with Multiple QoS Constraints

```
Begin
1.  $Q$ := 0;
2.  Generate a random invoking path $ip$;
3.  while evolution iteration $W$ is not reached do
4.   for each $t_i$ in $T$ do
5.     $WS_i$ := {}
6.     for each $ws_j$ in $S$ do
7.       if $ws_j$ is not candidate service of $t_i$ then continue
8.       else $WS_i$ := $WS_i$ + {$ws_j$}
9.     end for
10.    for each < $t_i$, $ws_{j,m}$> in $ip$ do
11.      Randomly generate triple < $c^m_{i,j}$, $r^m_{i,j}$, $e^m_{i,j}$,>
12.      if < $c^m_{i,j}$,$r^m_{i,j}$,$e^m_{i,j}$,> satisfying the QoS constraints then
13.        Calculate $Cost(ip_m)$, $Exec(ip_m)$ and $Rep(ip_m)$
14.        Q' := Q' + $Cost(ip_m)$ + $Exec(ip_m)$ + $Rep(ip_m)$
15.        if Q' > Q then Q := Q'
16.        else
17.          Mutate $ip$ using swap-mutation with the probability
             of 0.5;
18.          go to step 3
19.        end if
20.      end if
21.  end for
22. end while
end.
```

IV. Experiments Evaluation and Analysis

A. Experimental Settings

In order to evaluate the performance of the proposed service composed algorithm, extensive experiments are conducted in practical platform. In our experiments, Web services were developed using IBM's Web Services Toolkit and deployed on a cluster of PCs. All PCs had the same configuration: Pentium IV 2.8 MHz with 2G RAM, Windows 2000, Java 2 Enterprise Edition, and Oracle XML Developer Kit. Host nodes are connected through 100Mbits/sec LAN. The target Web application in our experiments is an extended version of service-oriented numerical optimization project, which is deprived from the prototype that designed by University of Southampton [23]. In our experiments, QoS data is retrieved by the service execution engine in different ways depending on the QoS dimension. The service broker logs appropriate QoS information during task executions, and the availability is calculated based on the information that it records about the up and down time of each service.

We investigate the performance in two situations: In a static situation, the QoS of any Web service will not be changed during a given composite service execution, and services are able to execute the tasks successfully and in conformance with their expected QoS; In a dynamic environment, the QoS of services may undergo changes during the execution of a composite service, which means that existing component services may become unavailable, new component services with better QoS may become available, component services may not be able to complete the execution of tasks.

B. Performance Comparison

The first series of experiments aimed at evaluating the QoS of composite service executions in both static and dynamic environments. The performance of the proposed algorithm (Evolution-strategy based Service Composition Algorithm, ESCA) are compared with three classic composition policy, which includes Global Composition Optimization Algorithm [8] (GCOA), Local Composition Optimization Algorithm [16] (LCOA), and Agent-based Service Composition Algorithm [20] (ASCA). Four QoS measurements are all considered separately in our experiments, and experimental results are shown in Fig. 2 and Fig. 3.

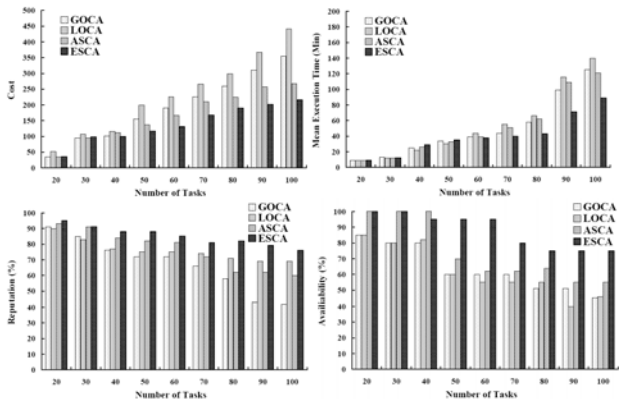

Fig. 2 QoS Performance Comparison in Static Environment

When the service composition occurs in static, the QoS of any Web service will not be changed during a given composite service execution, and services are able to execute the tasks successfully and in conformance with their expected QoS. So, QoS negotiations only are performed before the concrete service broker starts, which mean that re-negotiation never happens. In this case, we assume that reputation of a service is measured by its historic log that indicates the degree of its QoS compliance. As shown in Fig.2, the performance of GCOA are significantly better the LCOA policy. For example, the execution duration is consistently shorter when using GCOA than that of LCOA. It is because that the computation costs of GCOA is much higher that that of LCOA, which will be evaluated in the second experiment in details.

As to ASCA and ESCA, their performances are very similar when the number of tasks is less than 40. However, their variants become significant when we increase the task number of the experimental application. For example, the execution time of ESCA is about 25% shorter than that of ASCA when number of tasks is 100. As the analysis in [20], ASCA policy is based on the automotive characters of multiple agents, which is defined by administers of the system. When the size of an application is small, the ASCA can find an optimal solution by linear programming, which is similar to the proposed ESCA. However, when the number of tasks increases significantly, the solution of ASCA depends on the interaction of a large set of autonomous agents and the optimal solution can not be figured out any more.

When in dynamic environment, the QoS of services may undergo changes during the execution of a composite service, which means that existing component services may become unavailable, new component services with better QoS may become available. As shown in Fig. 3, the most differences between the two cases are that performance on reputation and availability measurements. The performance of both of QoS measurements are reduced about 12%~20% when in dynamic environments. Since the availability of Web services is dynamic, some services that are selected by the optimal execution plan may become unavailable when the task needs to be executed. Although the system can re-negotiate the unexecuted part of composite services, the executed part may become suboptimal, making the entire composite service execution suboptimal. There, execution time of a solution becomes uncertain, because re-negotiation will increased the delay of execution, but if better candidate service is available the benefits might compensate the delay brought by re-negotiation or re-optimization. So, in our experiments, we find that the performances of cost and execution time are almost the same whether in static environment or in dynamic environment.

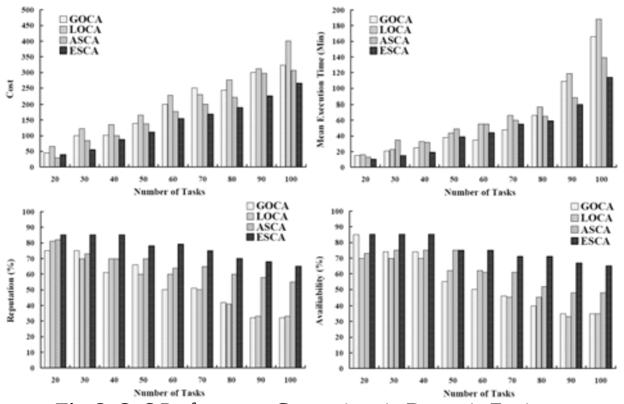

Fig. 3 QoS Performance Comparison in Dynamic Environment

By the above experimental results, we draw the conclusions as following: (1) The performance of ASCA and ESCA are better than GOCA and LOCA; (2) In presence of large-scale application, ESCA is more suitable than ASCA to find global optimal solution with multiple QoS constraints; (3) The stability of ESCA is better than the other three policies when in dynamic environment.

C. Computation Cost Comparison

In this experiment, we will investigate the computation cost of the proposed service composition algorithm. The aim is to provide a basis for determining the overhead for obtaining an optimal solution. For each test case, we executed the composite service 10 times and computed the average computation cost. Same as the first experiment, we choose the GCOA, LCOA and ASCA for comparison. As the complexity of ESCA is related to the parameter *W* (Evolution Iteration Number), so, we conducted the ESCA algorithm three with *W*=50, *W*=100, *W*=200. It is clear that bigger *W* will result in more computation cost but also bring about more optimal solution for ESCA. The experimental results are shown in Fig. 4 and Fig. 5.

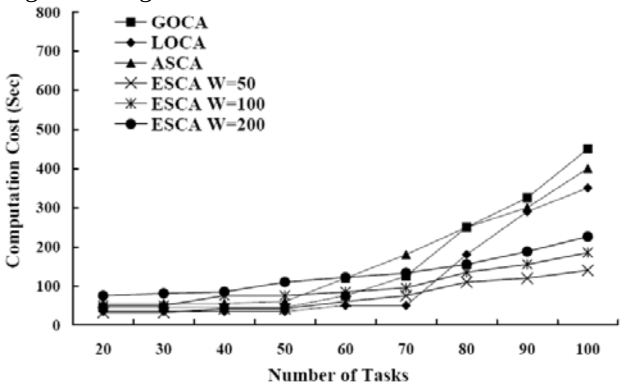

Fig. 4 Comparison of Computation Costs in Static Environment

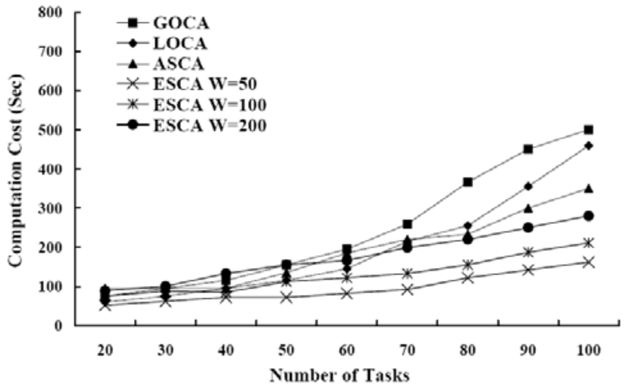

Fig. 5 Comparison of Computation Costs in Dynamic Environment

As shown in Fig. 4 and Fig. 5, the mean computation cost of GCOA is the highest, and LCOA is the second highest. It is especially true when the number of task become very large. In addition, we noticed that computation costs of GCOA and LCOA increases faster than that of ESCA and ASCA. It is because that implementation of GCOA and LCOA requires travailing the abstract service graph and candidate service set several time before a concrete service composition is obtained. The computation cost of global planning by exhaustive searching is very high even in very small scale in aspect of the number of tasks and candidate service. So the computation costs of both of the two composition policies are multi-linear related to the size of the target graph of an application. In addition, when in dynamic environment, re-negotiation often results in the change of candidate service set, which in turn will increase the computation cost of the two policies. The computation cost of ASCA policy is better than GCOA and LCOA, and the reason have been mentioned in the first experiments.

As to ESCA policy, the experimental results indicate that its computation cost does not increasing linearly with the increasing of the number of tasks. By analyzing the detailed experimental data, we noticed that most of optimal solution is obtain before the evolution iteration is increased to the maximum account. That is why the computation cost of ESCA is not linear to the *W* value, for instance, when *W* value is increased 100% then computation cost of ESCA only increased about 30%~45%. It is noteworthy that communication delay of QoS negotiation is not taken into account when calculating the computation cost of these policies, since this overhead is heavily depended on the network traffic and topology of the tested environment. More detailed study on negotiation delay when compositing large-scale application will be our next work.

The summary of this experiment is as following: (1) For those composition policies that based on exhausted searching technique, such as GOCA and LOCA, its extensibility will become a performance bottleneck in presence of large-scale distributed applications; (2) Both ESCA and ASCA are extensible service composition algorithms; (3) The computation cost of ESCA is less-linear related to *W* parameter, and a suitable value of *W* should be decided by the user's QoS requirements.

V. Conclusion

To address the issue of Web service composition with multiple QoS constraints, we introduce a service composition algorithm which formulates the service composition problem as a Multiple choice Multiple dimension Knapsack Problem and applies evolution strategy to obtain an approximate solution. Extensive experiments are conducted on a set of practical applications. Experimental results compare our method with other solutions and demonstrate the effectiveness of our approach toward the identification of an optimal solution to the QoS constrained Web service selection problem. In presence of large-scale application, the proposed ESCA algorithm is more suitable than ASCA to find global optimal solution with multiple QoS constraints. Also, the stability of ESCA is better than the other three policies when in dynamic environment. At present, the implementation of ESCA only considers four QoS measurements. Our future work will focus on more QoS requirement of business applications. Meanwhile, we will take efforts to improve the performance of QoS negotiation in our service composition framework.

Acknowledgment

This work was supported by a grant from the National Natural Science Foundation of China (No. 60673165 and No. 60970038). It is also a project supported by Scientific Research Fund of Hunan Provincial Education Department (No. 09c270). The authors sincerely thank Ms. Lee Xi in Chicago University for the correction of English writing.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*. Springer Verlag, 2003.

[2] T. Andrews and F. Curbera, “Business Process Execution Language for Web Services (version 1.1),” http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf, 2003.

[3] A. Ankolekar, et al., “DAML-S: Web Service Description for the Semantic Web,” *Proc. First Int’l Semantic Web Conf. (ISWC 02)*, 2002.

[4] D. Ardagna and B. Pernici, “Global and Local QoS Guarantee in Web Service Selection,” *Proc. Business Process Managment Workshop (BPM ’05)*, pp. 32-46, 2005. Int’l J. Business Performance Managment, vol. 1, no. 4, pp. 233-243, 2005.

[5] G. Bartoli, et al. “An Optimized Resource Allocation Scheme Based on a Multidimensional Multiple-Choice Approach with Reduced Complexity”, *Proc. of IEEE International Conference on Communications (ICC)*, pp. 1-6, 2011.

[6] J. Cardoso and A. Sheth, “Semantic E-Workflow Composition,” *J. Intelligent Information Systems*, vol. 21, no. 3, pp. 191-225, 2003.

[7] S. Ceri, F. Daniel, M. Matera, and F. Facca, “Model-Driven Development of Context-Aware Web Applications,” *ACM Trans. Internet Technology*, vol. 7, no. 2, 2007.

[8] L.A.G. Dacosta, P.F. Pires, and M. Mattoso, “Automatic Composition of Web Services with Contingency Plans,” *Proc. Int’l Conf. on Web Services Workshop (ICWS ’04)*, 2004.

[9] A.B. Hassine, S. Matsubara, and T. Ishida, “A Constraint-Based Approach to Horizontal Web Service Composition,” *Proc. Fifth Int’l Semantic Web Conf. (ISWC)*, pp. 130-143, 2006.

[10] S. Hwang et al., “Dynamic Web Service Selection for Reliable Web Service Composition,” *IEEE Trans. Services Computing*, vol. 1, no. 2, pp. 104-116, Apr.-June 2008.

[11] M.I. Islam and M.M. Akbar, “Heuristic algorithm of the multiple-choice multidimensional knapsack problem (MMKP) for cluster computing”, *Proc. of International Conference on Computers and Information Technology (ICCIT'09)*, pp.157-161, 2009.

[12] Q.A. Lang, “AND/OR Graph and Search Algorithm for Discovering Composite Web Services,” *J. Web Services Research*, vol. 2, no. 4, pp. 46-64, 2005.

[13] A. Lazovik, M. Aiello, and M. Papazoglou, “Associating Assertions with Business Proesses and Monitoring Their Execution,” *Proc. Int’l Conf. Service Oriented Computing (ICSOC ’04)*, pp. 94-104, 2004.

[14] Z. Maamar, S.K. Mostefaoui, and H. Yahyaoui, “Toward an Agent-Based and Context-Oriented Approach for Web Services Composition,” *IEEE Trans. Knowledge and Data Eng*., vol. 17, no. 5, pp. 686-697, May 2005.

[15] S. McIlraith and T.C. Son, “Adapting Golog for Composition of Semantic Web Services,” *Proc. Int’l Conf. Principles of Knowledge Representation and Reasoning (KRR)*, pp. 482-496, 2002.

[16] S. Oh, D. Lee, and S.R.T. Kumara, “Web Service Planner (WsPr): An Effective and Scalable Web Service Composition Algorithm,” *J. Web Services Research*, vol. 4, no. 1, pp. 1-23, 2007.

[17] A. Patil et al., “METEOR-S Web Service Annotation Framework,” *Proc. 13th Int’l Conf. World Wide Web (WWW)*, pp. 553-562, 2004.

[18] S. Patil and E. Newcomer, “ebXML and Web Services,” *IEEE Internet Computing*, vol. 7, no. 3, pp. 74-82, 2003.

[19] J. Schaffner, H. Meyer, and C. Tosun, “A Semi-Automated Orchestration Tool for Service-Based Business Processes.” *Proc. Second Int’l Workshop Eng. Service-Oriented Applications: Design and Composition (WESOA)*, pp. 50-61, 2006.

[20] H Tong, et al., “A Distributed Algorithm for Web Service Composition Based on Service Agent Model”, *IEEE Trans. on Parallel and Distributed Systems*, vol.2, no.12, pp.2008-2021, 2011.

[21] D.C. Vanderster, N.J. Dimopoulos, R.J. Sobie, “Metascheduling Multiple Resource Types using the MMKP”, *Proc. of IEEE/ACM International Conference on Grid Computing*, pp.231-237, 2006.

[22] S. Wang, W. Shen, and Q. Hao, “An Agent-Based Web Service Workflow Model for Inter-Enterprise Collaboration,” *Expert Systems with Applications*, vol. 31, no. 4, pp. 787-799, 2006.

[23] G. Xue, W. Song, S.J. Cox, A. Keane. Numerical Optimisation as Grid Services for Engineering Design. *Journal of Grid Computing*, vol.2, no.3, pp.223-238, 2004.

[24] T. Yu and K. Lin, “Service Selection Algorithms for Web Services with End-to-End QoS Constraints,” *Information Systems and E-Business Management*, vol. 3, no. 2, pp. 103-126, 2005.

Changsong Liu was born in 1974. He received his master degree in Xian University of Technology in 2004. Now, he is a Ph.D candidate in Central South University, and currently works in Hunan Institute of Engineering as lecturer. His research interests include service computing, quality of service management, distributed intelligence.

Dongbo Liu was born in 1974. He received his master degree in Jiangsu University of in 2004. He obtains the Ph.D degree in Hunan University in 2010, and currently works in Hunan Institute of Engineering as associate professor. His research interests include Web service, distributed intelligence, cloud computing and etc.

Ning Han was born in 1981. He received his bachelor degree in Beijing University of Science and Technology, and now persuading master degree in Xiangtan University. Currently, he works in the HP High Performance Lab of Hunan Institute of Engineering as a senior networking engineer. His research interests include complex networking deployment, distributed computing, information security technology, fault-tolerance in distributed systems.