Wu BY, Chi CH, Xu SJ *et al.* QoS requirement generation and algorithm selection for composite service based on reference vector. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(2): 357–372 Mar. 2009

QoS Requirement Generation and Algorithm Selection for Composite Service Based on Reference Vector

Bang-Yu Wu[1,2,3] (吴邦欲), Chi-Hung Chi[1,2] (支志雄), Shi-Jie Xu[1,2] (徐世杰), Ming Gu[1,2] (顾　明) and Jia-Guang Sun[1,2] (孙家广)

[1] *School of Software, Tsinghua University, Beijing 100084, China*

[2] *Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, China*

[3] *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*

E-mail: wby03@mails.tsinghua.edu.cn; {chichihung; guming; sunjg}@tsinghua.edu.cn; shijie_xu@163.com

Received May 7, 2008; revised November 18, 2008.

Abstract Under SOA (Service-Oriented Architecture), composite service is formed by aggregating multiple component services together in a given workflow. One key criterion of this research topic is QoS composition. Most work on service composition mainly focuses on the algorithms about how to compose services according to assumed QoS, without considering where the required QoS comes from and the selection of user preferred composition algorithm among those with different computational cost and different selection results. In this paper, we propose to strengthen current service composition mechanism by generation of QoS requirement and its algorithm selection based on the QoS reference vectors which are calculated optimally from the existing individual services' QoS by registry to represent QoS overview about the best QoS, the worst (or most economical) QoS, or the average QoS of all composite services. To implement QoS requirement, which is determined according to QoS overview, this paper introduces two selection algorithms as two kinds of experiment examples, one aiming at the most accurate service selection and the other chasing for trade-off between selection cost and result. Experimental results show our mechanism can help the requester achieve his expected composite service with appropriate QoS requirement and customized selection algorithm.

Keywords service-oriented architecture, service composition, quality of services, service selection

1 Introduction

Service-oriented architecture (SOA) promises on-demand creation of new services which are composed of multiple component services. There has been substantial amount of previous research related to the topic of service composition. Related standards and reference architectures are proposed. Languages and protocols such as SOAP, WSDL, UDDI[1], BPEL4WS[2], ebXML[3] are available to provide infrastructure support for Web service composition. Below are some of the major challenges in the service composition process.

Firstly, according to the service design principles, services should be loosely coupled and autonomous. Together with the fact that services are published before requesters (with their requirements) come for selection, individual component services should be independent of each other. However, in a composite service, this independency disappears as all the components involved are aggregated together to achieve the same overall goal, which brings about much work to do[4−7]. There might exist many possible ways of aggregating component services together to form the required composite service of a given set of functionalities. With an expected increase in the number of component services available in the service registry, service composition algorithm is being confronted with a large number of combination choices. This problem has attracted many researches.

Secondly, before service selection, the required QoS should be specified. However, much work of service selection assumes that QoS requirement already exists, and its generation is ignored. By the definition of "software service", each user should have his own unique QoS requirement and constraint. For example, some users might focus on performance issues such as latency, whereas others might demand more on reliability

Regular Paper

Supported by the National Natural Science Foundation of China under Grant No. 90604028, the National Basic Research 973 Program of China under Grant No. 2004CB719406, and the National High-Tech Research and Development 863 Program of China under Grant Nos. 2008AA01Z12 and 2007AA01-Z122.

of business transaction. But due to the independency and loose coupling of component services, the overview of QoS information of all possible composite services, such as the possible best services, the average services, or the worst (most economical) services is not known by a requester. Therefore, how to generate an appropriate QoS requirement for the requester is also a major problem.

Thirdly, although lots of service selection algorithms have been proposed, few papers discuss how to bind QoS requirement with the algorithms. Since different algorithms will consume different computational costs, resulting in different selection results. For example, as getting the "best quality" services or "good enough quality" requires different algorithms, a corresponding mechanism should be designed for requesters to pick up his preferred algorithm to reach his required QoS.

The aim of this paper is to strengthen current service composition mechanism by providing generation of QoS requirement and its implementation algorithm between the requester and registry. Firstly, given a workflow by a requester, lots of composite services with different QoS composition could be constructed by component services with QoS information published in registry. If the requester wants to propose really practical QoS, he could negotiate with the registry to investigate the overall QoS composition information of all possible composite services until he understands what QoS or what kind of service he needs. Secondly, the registry negotiates with the requester about which algorithm to select from the algorithms stored in registry, followed by executing service selection. Therefore, the QoS requirement generation, together with the algorithm customization, lets the requester fulfill an approving composite service for a given workflow. To implement the target, this paper presents an idea of QoS reference vector and improves SOA framework.

The outline for the rest of this paper is as follows. In Section 2, related work on service composition is surveyed. In Section 3, we briefly describe the workflow of composite service, and the improved SOA framework to support QoS requirement generation. Section 4 discusses the general service of the QoS model and calculation methods of the best, worst and average composed QoS for different individual metrics. Section 5 presents a mechanism and computation methods to obtain the QoS requirement for a composite service based on QoS reference vector. Section 6 presents the implementation algorithm selection, two service selection algorithms are introduced as examples: one for the most suitable QoS selection with higher execution overhead and the other for fast service selection through effective trade-off between the overall quality of composite service and service selection overhead. Experimental study of these two algorithms is presented in Section 7. Finally, the paper concludes in Section 8.

2 Related Work

Web service composition involving QoS has received much attention during the recent years. [8] addresses service composition based on workflow. Its model does not support parallelism nor branching. It just defines a composite service as a chain of service operations. WebQ[9] conducts adaptive selection process and simultaneously provides binding and execution for workflow. However, their QoS selection only considers service load and makes decisions according to individual component service. A QoS-driven selection algorithm during the execution of a composite service is given in [10], it considers multiple QoS criteria such as price, duration, and reliability. [11] proposes to identify the best set of services at runtime by using a mixed integer linear programming. [12] formalizes three kinds of optimal service selection problems based on different criteria and studies their complexity and implement solutions. Evaluating on composition algorithms[13] points out that service composition is a multi-dimensional optimisation problem which results in an exponential computation effort for computing an optimal solution. It is similar to other combinatorial problems — the knapsack problem and the resource constraint project scheduling problem, and several possible heuristics are described for these problems and their efficiency. An approach to triggering and performing composite service replanning is proposed during execution when the actual QoS values deviate from the QoS requirement, or the execution path may not be the foreseen[14]. However, much of the work focuses on composition algorithm, without considering how to help the requester create his QoS requirement, which is the premise to do composition. Furthermore, they rarely touch the idea of customizing preferred selection algorithm to implement required QoS for the requester. However, the general algorithms of integer programming and exhaustive search used in [10, 11] are also introduced as our experimental example.

Some recent proposals face the QoS generation problem. To solve QoS issues related to security, reliability, and performance when implementing "write-once, reuse everywhere" for component software, authors in [15] present a middleware-based approach to managing dynamically the changing of QoS requirements of components, they provide middleware enhancements to match, interpret, and mediate QoS requirements of

clients and servers at deployment time or runtime. QoS parameters generation is done by asking the service providers to improve their QoS, when a feasible composite solution does not exist, it demands requester and provider find a new agreement on QoS parameters[11]. [16] realizes the need of QoS requirement generation, and it presents an agent-based coordinated-negotiation generation framework. The basic idea is mapping the overall QoS requirements onto component service QoS requirements, i.e., QoS requirements for each service type within the composition, then negotiating with potential service providers. [17] suggests a policy driven and multi-agent generation approach to achieving win-win in the composition and cooperation of the services according to the rules. The methodology is introduced in [18] considering how to model a service composition as a constraint satisfaction problem and how to solve the problem by a multi-agent negotiation generation algorithm. However, the major work relies on providers or the agents of the provider to involve in generation of the individual QoS that each component service should provide. The goal of our approach is generating QoS requirement for the requester instead of the provider about the overall QoS of composite service.

All the work contributes much to service composition. However, our approach can be seen as complementary to them. We propose to generate the required QoS for users, which is assumed to already exist in above researches, we also advise to provide a mechanism for a requester to determine which service selection algorithm he expects registry to use, the algorithm could be any designed by registry or others provided by the requester himself.

3 System Model

In this section, we describe the workflow of composite service and service QoS model, and improve SOA framework to support QoS requirement generation and algorithm selection. We assume that the QoS of each component service is relatively stable without needing to switch frequently among other services.

3.1 Workflow Model of Composite Service

We define three types of basic relationship among services in a typical service workflow: "Sequential" (type S, marked as "$\sim$"), "Parallel Exclusion" (type PE, marked as "$\backslash$"), and "Parallel And" (type PA, marked as "$\wedge$"). With S, services are executed in a sequential order. With PA, services should be executed in parallel. With PE, services are executed exclusively, only one parallel service responds to a request execution.

We construct a probabilistic request execution workflow model for composite service based on above three basic types of workflow. Note that although the given service workflow is fixed, its execution instance for a specific request is still dynamic due to the transition probability of requests among component services. That is, the workflow process is a stochastic process with a finite set of component services.

Given a composite service, its request execution workflow graph (REWG) is constructed recursively using the above three basic workflow types. To simplify the discussion, we assume that REWG is acyclic and PE services or PA services have only one direct predecessor (or initial vertex) and one direct successor (or final vertex). In REWG, a vertex represents a service or business process and an edge expresses the execution order or precedence constraint. Let graph $REWG = \langle V, E\rangle$ denote the workflow model, where V is a set of tasks, $V = \{t_1, t_2, \ldots, t_n\}$, and E is a set of directed edges, $E = \{\langle t_i, t_j\rangle | i \leqslant n,\ j \leqslant n,\ i < j\}$.

As an example, RUBiS Benchmark[19] implements an auction site prototype modeled after eBay. It includes some core tasks: selling, browsing, and bidding, etc. A composite service is formed only after the service selection for each task. In general, there exist multiple service providers to satisfy one task function, and the goal of service selection is to select the one which can meet the requester's QoS requirements.

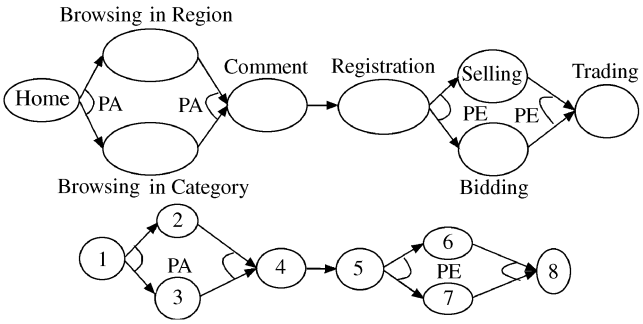


Fig.1. Workflow graph of RUBiS.

Fig.1 gives an example of RUBiS Benchmark workflow. It has eight vertices or tasks (later in our experimental study, we will use it as an example). Its formal expression is:

$$t_1 \sim (t_2 \wedge t_3) \sim t_4 \sim t_5 \sim (t_6 \backslash t_7) \sim t_8.$$

Definition 1 (Execution Path (EP)). *An execution path of a workflow consists of a sequence of vertices* $\{t_1, t_2, \ldots, t_n\}$*, its similar definition is presented*

in many researches, it represents an execution process of request in workflow, its connections follow the original workflow such that t_1 is the start vertex, t_n is the end vertex, and for every vertex t_i $(1 \leqslant i \leqslant n)$:

(1) *t_i is a direct successor to one or more of the vertices in $\{t_1, t_2, \ldots, t_{i-1}\}$;*

(2) *t_i is not a direct successor to any to the vertices in $\{t_{i+1}, t_{i+2}, \ldots, t_n\}$;*

(3) *There is no vertex t_j in EP $\{t_1, t_2, \ldots, t_i, \ldots, t_n\}$ such that t_i has the relationship of $t_i \backslash t_j$. That is to say, all PE vertices must belong to different EPs;*

(4) *If t_j belongs to EP $\{t_1, t_2, \ldots, t_i \ldots, t_n\}$, then any vertex t_i with the relationship of $t_i \wedge t_j$ also belongs to the EP. That is to say, all PA vertices must belong to the same EP.*

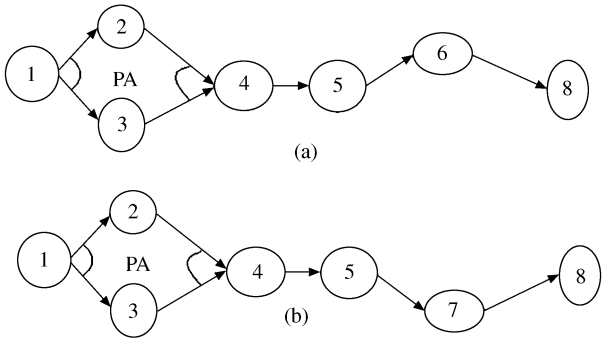


Fig.2. Two EPs of Fig.1.

During the execution process of a given workflow, the execution path might be uncertain at the beginning because of the vertices of type PE, which branches the execution flow. However, as the request traverses through the graph, its path becomes clearer. Finally, the execution path is formed entirely when requests execution are finished at the end service of workflow. An execution path is a subgraph of the original workflow graph with only two basic workflow types, type S and PA. There might be multiple EPs in one workflow. Any request must be serviced along one and only one of these EPs. The EPs could be identified based on the traversal of the workflow graph. Fig.2 shows two EPs in the workflow of Fig.1. Their formal expressions are: (a) $t_1 \sim (t_2 \wedge t_3) \sim t_4 \sim t_5 \sim t_6 \sim t_8$; and (b) $t_1 \sim (t_2 \wedge t_3) \sim t_4 \sim t_5 \sim t_7 \sim t_8$. In this paper, we also assume that for each task t_i, there are m candidate services to be selected, and they are denoted by $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,m}\}$.

3.2 Improving SOA Framework

The approach to QoS requirement generation and implementation algorithm selection proposed relies on the definition to improve SOA framework where QoS information published by service provider is trustful, and requester is able to provide composition algorithm to the registry which is used for his service composition. The improved SOA framework is depicted in Fig.3.

Let us show how the process of service composition can be enriched through the QoS requirement generation and implementation algorithm selection.

• Service publication: during service publication, the QoS information which will be used for future QoS computation is also provided through some specification, such as WSDL.

• Service composition: requester submits his composite service workflow structure according to his application logic to the registry for future service composition implementation; registry analyzes this workflow, starting to compute the composed QoS overview including the possible best QoS, the possible worst QoS, and

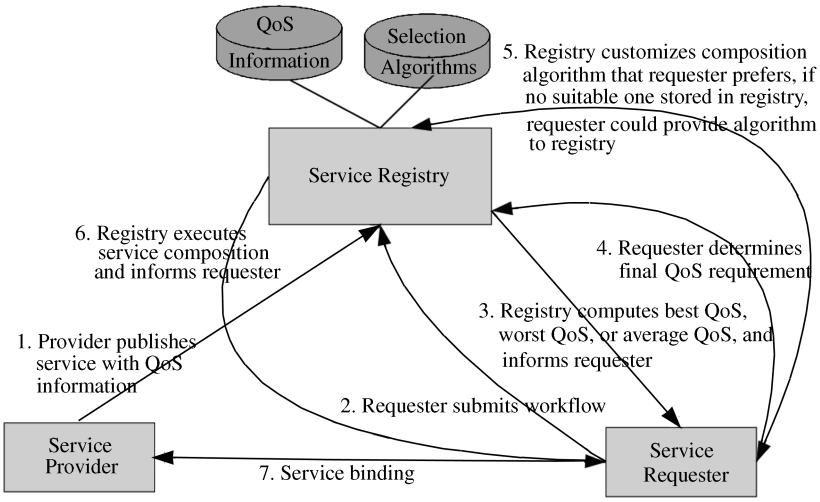


Fig.3. The improved SOA framework.

the average QoS based on individual QoS information and workflow. After getting the QoS overview for his workflow, the requester needs to determine which one is his preference. Does he require a best service or the most economical one, or the average one? If none, he will continue to determine his real required QoS by assigning certain impact factor to them, such as 80% of the best. During the process of QoS generation, the registry needs to compute the composed QoS for requester workflow. After generating the QoS, how to implement this QoS is related to the problem of composition algorithm. The registry may store multiple selection algorithms, each has its own characteristic, some maybe execute with fast speed but the result is not exactly accurate, some could get accurate result, but only suitable for small size environment. Therefore, the requester must determine his appropriate implementation algorithm, with considering the number of workflow tasks, the number of component services for each task, the QoS constraint etc. The requester could provide his own algorithm to registry to execute selection if he is satisfied with none in registry. Finally, registry begins to do service selection according to the generated QoS requirement and implementation algorithm.

• Service binding: this behavior is similar to traditional SOA, the component services should be invoked to execute.

4 General QoS Model and Composition Calculation

In this paper, we would like to propose a general QoS model. Items such as *price*, *time*, *reliability*, *security* and *trust* will be considered inside (just like most models published[10,20,21].) since these are common items for QoS. Of course our idea will work also for other QoS metrics. Any QoS element can be added into or deleted from the vector according to user requirements without affecting the validity and operation of our model. The second factor to choose these items is that they represent two kinds of criteria, one is positive such as trust, security and reliability. The bigger the value is, the better its QoS will be. The other is negative, such as time and price. They are just the opposite. The bigger the value is, the worse the QoS will be. The third factor is that they represent different computation methods of composing QoS. The price is the sum of all the services on workflow, the execution time depends on the critical path, the reliability is given by the product of reliabilities, the trust is the average trust, and the security is the minimum value of individual security levels. The computation method aims at the execution of the whole workflow instead of single request. EP is used to analyze the execution time.

Ideally, the best composite service requires each composed QoS is the best. Similarly, the worst or average composite service means that each composed QoS is the worst or average. So calculation of different QoS metrics for composite service is the base for generating QoS reference vector later.

In theory, if a given workflow has n tasks and each task has m candidate services, m^n possible composite services could be constructed, independent of the best, worst or average QoS values. During QoS computation, registry will keep computing the best or worst composed QoS metrics from m^n composite services. It is a real problem, especially when n is big. In this section, we focus on optimization calculation directly from the existing individual services' QoS instead of from the m^n composite services.

Without losing generality, let us define QoS matrix $\boldsymbol{U}$ to represent QoS values of all candidate service sets for any given QoS metrics. So there are k QoS matrices $\boldsymbol{U}$. Each column in $\boldsymbol{U}$ represents the QoS values of all candidate services for each task.

$$\boldsymbol{U}_{\text{QoS metrics}} = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ u_{m,1} & u_{m,2} & \cdots & u_{m,n} \end{bmatrix},$$

where, for example, in $\boldsymbol{U}_{\text{price}}$, $u_{i,j}$ represents the price of service $s_{i,j}$. For better presentation, we also define $f_{\text{price}}(s_{i,j})$ to present $u_{i,j}$ in $\boldsymbol{U}_{\text{price}}$, and similarly, $f_{\text{trust}}(s_{i,j})$ to present $u_{i,j}$ in $\boldsymbol{U}_{\text{trust}}$, etc.

4.1 Price

• The best:
$f_{\text{price}}(REWG) = \sum_{i=1}^{n}(\min\{u_{i,j}\}|\, j = 1, 2, \ldots, m)$.
• The worst:
$f_{\text{price}}(REWG) = \sum_{i=1}^{n}(\max\{u_{i,j}\}|\, j = 1, 2, \ldots, m)$.
• The average value depends on the sum of all composite services, and G_i is the i-th composite service to do the following:

$$f_{\text{price}}(REWG) = \frac{\sum_{i=1}^{m^n} f_{\text{price}}(G_i)}{m^n} = \frac{m^{n-1} \times \sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m^n}.$$

Because for each element $u_{i,j}$, it will appear in m^{n-1} composite services, we continue to have

$$f_{\text{price}}(REWG) = \frac{m^{n-1} \times \sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m^n} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m}.$$

The optimized computation complexity is $O(m \times n)$.

4.2 Reliability

Reliability refers to the probability of the system's availability that a requester can complete the service invocation instance on it successfully. In general, its value ranges from 0 to 1.

• The best:

$$f_{\text{relia}}(REWG) = \prod_{i=1}^{n}(\max\{u_{i,j}\}| \quad j = 1, 2, \ldots, m).$$

• The worst:

$$f_{\text{relia}}(REWG) = \prod_{i=1}^{n}(\min\{u_{i,j}\}| \quad j = 1, 2, \ldots, m).$$

• The average:

For QoS metrics such as reliability, which depends on the product of QoS elements, we should pick up an element from each column in matrix $\boldsymbol{U}$. So we have:

$$\begin{aligned} f_{\text{relia}}(REWG) &= \frac{\sum_{i=1}^{m^n} f_{\text{relia}}(G_i)}{m^n} \\ &= \frac{\sum_{i_n=1}^{m}\sum_{i_{n-1}=1}^{m}\cdots\sum_{i_1=1}^{m} u_{i_1,1}u_{i_2,2}\cdots u_{i_n,n}}{m^n} \\ &= \frac{\sum_{i_n=1}^{m} u_{i_n,n} \times \sum_{i_{n-1}=1}^{m} u_{i_n,n-1} \times \cdots \times \sum_{i_1=1}^{m} u_{i_1,1}}{m^n} \\ &= \frac{\sum_{i=1}^{m} u_{i,n} \times \sum_{i=1}^{m} u_{i,n-1} \times \cdots \times \sum_{i=1}^{m} u_{i,1}}{m^n} \\ &= \frac{\prod_{j=1}^{n}\sum_{i=1}^{m} u_{i,j}}{m^n}. \end{aligned}$$

Therefore, getting the average value of reliability that is dependent on their product is very simple: (i) finding the QoS sum of all candidate services by $\sum_{i=1}^{m} u_{i,j}$, (ii) calculating the product of all sums, and (iii) computing its average value. The optimized computation complexity of this process is $O(m \times n)$.

4.3 Trust

Trust measures a service's trustworthiness. Trust boosts user confidence and facilitates judgment about service reputations. In general, we set the value of trust ranging from 1 to 5.

• The best:

$$f_{\text{trust}}(REWG) = \sum_{i-1}^{n}(\max\{u_{i,j}\}|j = 1, 2, \ldots, m)/n.$$

• The worst:

$$f_{\text{trust}}(REWG) = \sum_{i-1}^{n}(\min\{u_{i,j}\}|j = 1, 2, \ldots, m)/n.$$

• The average:

$$\begin{aligned} f_{\text{trust}}(REWG) &= \frac{\sum_{i=1}^{m^n} f_{\text{trust}}(G_i)}{m^n} = \frac{m^{n-1} \times \sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m^n \times n} \\ &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m \times n}. \end{aligned}$$

The optimized computation complexity is $O(m \times n)$.

4.4 Security

Security focuses on protecting users and businesses from anonymous intrusions, attacks, vulnerabilities, etc. Security is measured by the standard security level which is instituted by authoritative organizations or government. Examples of such security level standards include ITSEC[22] and CC[23]. We set the value of security level ranging from 1 to 5 in our experiment. In this paper, we take the general assumption that the final security level of a composite service is determined by the minimum value of individual security levels of its component services.

• The best:

$$f_{\text{sec}}(REWG) = \min\{\{\max\{u_{i,j}\}|j = 1, 2, \ldots, m\}| \\ i = 1, 2, \ldots, n\}.$$

• The worst:

$$f_{\text{sec}}(REWG) = \min\{\{\min\{u_{i,j}\}|j = 1, 2, \ldots, m\}| \\ i = 1, 2, \ldots, n\}.$$

• The average:

For the QoS metrics such as security, which depends on extreme values, do the following:

$$f_{\text{sec}}(REWG) = \frac{\sum_{i=1}^{m^n} f_{\text{sec}}(G_i)}{m^n} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j} \times x_{i,j}}{m^n},$$

where x_{ij} is the occurrence frequencies of $u_{i,j}$ to be selected as $f_{\text{sec}}(G_i)$, because the result of $f_{\text{sec}}(G_i)$ is one of $m \times n$ elements in the matrix $\boldsymbol{U}$, and we just need to find out that how many times each element is selected as $f_{\text{sec}}(G_i)$.

The next problem to be solved is to find $x_{i,j}$. We observe that whether the element $u_{i,j}$ is selected relies on how many elements in each column in $\boldsymbol{U}$ are bigger than itself. The combination number of these elements is just the value of $x_{i,j}$. Assume that y_i elements in column i are less than $u_{i,j}$, then $x_{i,j} = \prod_{i=1,i\neq j}^{n} y_i$. In this way, the security element could be found.

$$f_{\text{sec}}(REWG) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j} \times x_{i,j}}{m^n} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(u_{i,j} \times \prod_{i=1,i\neq j}^{n} y_i\right)}{m^n}.$$

If the value of $u_{i,j}$ is unique, $x_{i,j}$ could be calculated easily. However, if the value of elements in other column is the same as $u_{i,j}$, the problem will become harder. It is because the repeated elements will result in incorrect $x_{i,j}$. Therefore, we introduce an infinitely small number ε to remove these repeated elements. Consequently, we first find the occurrence frequency of the value of each element appears in $\boldsymbol{U}$. Then, we add different multiples o ε to these repeated elements. These two steps are formulated as follows:

(i) Producing tag matrix about occurrence frequencies of elements with the same value existing in $\boldsymbol{U}$. If elements with the same value appear in $\boldsymbol{U}$ for some occurrence frequency, let $u_{i,j}^{a,k}$ represent a (i.e., $u_{i,j} = a$), and that $u_{i,j}$ is the k-th time the value a appears.

$$\boldsymbol{T} = \begin{bmatrix} u_{1,1}^{a,1} & u_{1,2}^{d,1} & \cdots & u_{1,n}^{b,1} \\ u_{2,1}^{c,1} & u_{2,2}^{a,2} & \cdots & u_{2,n}^{c,2} \\ \vdots & \vdots & \vdots & \vdots \\ u_{m,1}^{c,3} & u_{m,2}^{b,2} & \cdots & u_{m,n}^{a,k} \end{bmatrix},$$

$$\boldsymbol{U}^* = \begin{bmatrix} u_{1,1}^* & u_{1,2}^* & \cdots & u_{1,n}^* \\ u_{2,1}^* & u_{2,2}^* & \cdots & u_{2,n}^* \\ \vdots & \vdots & \vdots & \vdots \\ u_{m,1}^* & u_{m,2}^* & \cdots & u_{m,n}^* \end{bmatrix}.$$

(ii) With reference to u_{ij}^{ak} in $\boldsymbol{T}$, updating $\boldsymbol{U}$ in this way: $u_{i,j}^* = u_{i,j} + (k-1)\varepsilon = a + (k-1)\varepsilon$.

After these two steps, a QoS matrix $\boldsymbol{U}^*$ without repeated elements is produced. For any two elements in $\boldsymbol{U}^*$ that are different, we have:

$$\begin{aligned} f_{\text{sec}}(REWG) &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(u_{i,j}^* \times x_{i,j}^*\right)}{m^n} \\ &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(u_{i,j}^* \times \prod_{i=1,i\neq j}^{n} y_i^*\right)}{m^n} \\ &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left((u_{i,j} + (k-1)\times\varepsilon) \times \prod_{i=1,i\neq j}^{n} y_i^*\right)}{m^n} \\ &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\left(u_{i,j} \times \prod_{i=1,i\neq j}^{n} y_i^*\right)}{m^n}. \end{aligned}$$

The optimized computational complexity here is $O((m \times n)^2)$.

4.5 Time

Time is the average execution time determined by the critical paths among all the possible EPs. Let a be the number of EPs, and $f_{\text{time}}(EP_k)$ be the execution time of the i-th EP.

- The best:

$$f_{\text{time}}(REWG) = \sum_{i=1}^{a} f_{\text{time}}(EP_k)/a,$$

where $f_{\text{time}}(EP_k) = \sum_{i=1}^{n}(\min\{u_{i,j}\}|j = 1, 2, \ldots, m)$, and $s_{i,j}$ locates in the critical path of EP_k.

- The worst:

$$f_{\text{time}}(REWG) = \sum_{i=1}^{a} f_{\text{time}}(EP_k)/a,$$

where $f_{\text{time}}(EP_k) = \sum_{i=1}^{n}(\max\{u_{i,j}\}|j = 1, 2, \ldots, m)$ and $s_{i,j}$ locates in the critical path of EP_k.

- The average:

Among our QoS model, time the most complex QoS metrics, its computation depends on the critical path in each EP.

Different service combination results in different critical path. If we find all the critical paths, sum up the time for each path, and then find their average value, this process will be an NP-hard problem, because there are m^n composite services.

To solve this problem, we first need to define some terms. For vertices that appear on each critical path, we label them as *public vertices*. Let us assume there are m candidates for each public vertex. The vertices

with type PA between two public vertices are defined as *super vertices*. Assume that there are i vertices constructing a super vertex, then there will be m^i candidates for this super vertex. As a result, EP is mapped onto a vertex sequence of public vertices and super vertices. The critical path is determined by the value of these two kinds of vertices. And the values of public vertices are assured to contribute to the critical path while the values of super vertices depend on the critical vertices.

Theorem 1. *For each EP with the vertex sequence being composed of public vertices and super vertices $\langle R_1, R_2, R_3, \ldots, R_z\rangle$, assume the numbers of their candidates are $l_1, l_2, l_3, \ldots, l_z$. Let vector $\boldsymbol{L}_i = \langle c_{1,i}, c_{2,i}, \ldots, c_{l_i,i}\rangle$ represent the time of S_i, then*:

$$\text{average time of EP} = \frac{\sum_{j_n=1}^{l_n}\sum_{j_{n-1}=1}^{l_n}\cdots\sum_{j_1=1}^{l_1}\sum_{i=1}^{z} c_{j_i,i}}{\prod_{j=1}^{z} l_j} = \sum_{i=1}^{z}\left[\frac{\sum_{j=1}^{l_i} c_{j,i}}{l_i}\right].$$

Proof. The time QoS matrix of all the candidates for sequence vertices $\langle R_1, R_2, R_3, \ldots, R_z\rangle$ is given below as $\boldsymbol{L}$:

$$\boldsymbol{L} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,z} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,z} \\ \vdots & \vdots & \vdots & \vdots \\ c_{l_1,1} & c_{l_2,2} & \cdots & c_{l_z,z} \end{bmatrix},$$

time of $EP = \sum_{\forall j_i\in\{1,2,3,\ldots,l_i\}}^{i=1} c_{j_i,i}$, and average time of EP =

$$\frac{\sum_{j_n=1}^{l_n}\sum_{j_{n-1}=1}^{l_{n-1}}\cdots\sum_{j_1=1}^{l_1}\sum_{i=1}^{z} c_{j_i,i}}{\prod_{j=1}^{z} l_j} = \frac{\sum_{j_n=1}^{l_n}\sum_{j_{n-1}=1}^{l_{n-1}}\cdots\sum_{j_1=1}^{l_1}(c_{j_1,1}+c_{j_2,2}+\cdots+c_{j_z,z})}{l_1\times l_2\times\cdots\times l_z}.$$

The above equation has the meanings of picking up any element from each vector in $\boldsymbol{L}$ in order, summing them up, and then averaging all the sums. Therefore, for any element $c_{j,i}$, it will appear $l_1\times l_2\times\cdots\times l_{i-1}\times l_{i+1}\times\cdots\times l_n$ times. Based on this analysis, we have the following equation:

$$\text{average time of EP} = \frac{1}{l_1\times l_2\times\cdots\times l_z}\Big\{\sum_{i=1}^{z}[(l_1\times l_2\times l_{i-1}\times l_{i+1}\times\cdots\times l_z)\times(c_{1,i}+c_{2,i}+\cdots+c_{l_i,i})]\Big\}$$

$$= \sum_{i=1}^{z}\Big[\frac{(c_{1,i}+c_{2,i}+\cdots+c_{l_i,i})}{l_i}\Big] = \sum_{i=1}^{z}\left[\frac{\sum_{j=1}^{l_i} c_{j,i}}{l_i}\right].$$

Thus, Theorem 1 is proved. □

Theorem 1 shows that we can compute the average critical path by finding the average value of candidates for each vertex along $\langle R_1, R_2, R_3, \ldots, R_z\rangle$ through $\frac{\sum_{j=1}^{l_i} c_{j,i}}{l_i}$, where $l_i = m$ for each public vertex, and $l_i = m^j$ for each super vertex with j vertices with type PA. For public vertex, $c_{j,i}$ is known; for super vertex, $c_{j,i}$ can be computed using the above method. The computation complexity of finding the average time of EP is $O((m\times n)^2 + m\times n)$. From the above formula, we see that the fewer the super vertices is, the faster the computation process will be. In particular, it will be the fastest if REWG is just a simple sequential composite flow.

The average time of composite service could be computed based on the average time of EP. Assume a EPs could be decomposed from REWG and there are p_i vertices in EP_i, then the number of composite services supported by EP_i is $\lambda_i = m^{p_i}$.

$$f_{\text{time}}(REWG) = \Big(\sum_{i=1}^{a}(\text{average time of } EP_i\times\lambda_i)\Big)\Big/\Big(\sum_{i=1}^{a}\lambda_i\Big).$$

The computation complexity of this step is $O(n)$.

Based on the above analysis, the QoS vector of composite service is defined as:

$$f(REWG) = \langle f_{\text{price}}(REWG), f_{\text{time}}(REWG), f_{\text{relia}}(REWG), f_{\text{trust}}(REWG), f_{\text{sec}}(REWG)\rangle.$$

The individual QoS vector for each component service $s_{i,j}$ is presented as: $\langle f_{\text{price}}(s_{i,j}), f_{\text{time}}(s_{i,j}), f_{\text{relia}}(s_{i,j}), f_{\text{trust}}(s_{i,j}), f_{\text{sec}}(s_{i,j})\rangle$.

5 Generation of QoS Requirement for Composite Service

For most service composition algorithms, one of the most critical assumptions is that the required QoS requirement is ready beforehand. However, it is difficult for the requester to determine his required SLA, because he has limited knowledge about all the possible composed QoS without information from registry.

To solve this problem, we introduce the concept of required QoS reference vector (***CRV***) to composite service through which a requester can express his required QoS. To generate ***CRV***, three other related QoS reference vectors need to be negotiated and presented to users. They are the best QoS reference vector (***CHV***), worst QoS reference vector (***CLV***), and average QoS reference vector (***CAV***). With the necessary QoS knowledge, the requester can determine his own ***CRV***, then registry uses it to do selection.

To explain our idea based on reference vector conveniently, Table 1 lists assumed QoS of two candidate services for each task in Fig.1. The candidate service set is $\boldsymbol{S}_i = \{s_{i,1}, s_{i,2}\}$, the target of composing service is to select one suitable service from $\boldsymbol{S}_i$. The values in Table 1 are adopted as examples in the following.

Table 1. Assumed QoS of Candidate Service

Service		QoS				
		Price	Time	Reliability	Trust	Security
S_1	s_{11}	8	5	0.92	4	2
	s_{12}	12	4	0.98	5	2
S_2	s_{21}	9	8	0.90	3	2
	s_{22}	13	6	0.98	4	3
S_3	s_{31}	10	9	0.94	3	2
	s_{32}	12	7	0.98	5	3
S_4	s_{41}	11	5	0.93	3	2
	s_{42}	15	4	0.98	4	2
S_5	s_{51}	8	5	0.94	4	3
	s_{52}	13	4	0.98	5	3
S_6	s_{61}	10	5	0.95	4	3
	s_{62}	14	4	0.98	5	2
S_7	s_{71}	11	6	0.95	4	3
	s_{72}	16	5	0.98	5	2
S_8	s_{81}	12	9	0.96	4	3
	s_{82}	18	7	0.98	5	3

Before we go to the discussion, let us define the QoS matrix of candidate services set $\boldsymbol{S}_i$ for each task as follows, and assume there are k items in QoS model.

$$\boldsymbol{Q}^x = \begin{bmatrix} q^x_{1,1} & q^x_{1,2} & \cdots & q^x_{1,k} \\ q^x_{2,1} & q^x_{2,2} & \cdots & q^x_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ q^x_{m,1} & q^x_{m,2} & \cdots & q^x_{m,k} \end{bmatrix},$$

where $q^x_{i,j}$ is the j-th QoS item of candidate service $s_{x,i}$ for task t_x, so there are: $\boldsymbol{Q}^1, \boldsymbol{Q}^2, \boldsymbol{Q}^3, \ldots, \boldsymbol{Q}^n$, for each task in REWG.

In fact, $\boldsymbol{Q}^x$ matrix has the same meaning with $\boldsymbol{U}_{\text{QoS metrics}}$ matrix, both of their elements represent the value of $f_{\text{QoS metrics}}\ (s_{i,j})$, but each is suitable for presentation conveniently in different context.

5.1 Computation of the Possible Best QoS

In general, when a requester consults the registry about what is the best composite service it could provide, registry will compose services with the best QoS metrics for each task. For example, select the fastest services on the critical path and sum up all times involved to be the minimum execution time, then inform the result to the requester. We define ***CHV*** as the best QoS. ***CHV*** investigates the possible best composite service, which is the most perfect. This ideal composite service may never exist since it requires each QoS metrics to be the best. However, it helps the requester understand what the best is, then specifies his required SLA by referencing ***CHV***.

Definition 2. ***CHV*** $= \langle chv_1, chv_2, \ldots, chv_k \rangle$, *where* $chv_i = f_{QoSmetric\,i}(REWG)$, *(with best*$\{q^1_{j,i} | 1 \leqslant j \leqslant m\}$, *best*$\{q^2_{j,i} | 1 \leqslant j \leqslant m\}, \ldots,$ *best*$\{q^n_{j,i} | 1 \leqslant j \leqslant m\}$*), and best*$=$ max *(or min). The bigger (or smaller)* $q^x_{j,i}$ *is, the better the QoS will be. For example, both higher reliability and shorter time mean better QoS.*

According to the calculation method of best QoS for each metrics in Section 4, the ***CHV*** of five common QoS metrics for Fig.1 with the assumed QoS in Table 1 is given as:

$$\begin{aligned} f_{\text{price}}(REWG) &= \sum_{i=1}^{8} (\min\{f_{\text{price}}(s_{i,j}) | j = 1, 2\}) = 79; \\ f_{\text{trust}}(REWG) &= \sum_{i=1}^{8} (\max\{f_{\text{trust}}(s_{i,j}) | j = 1, 2\})/8 \\ &= 4.7; \\ f_{\text{time}}(REWG) &= \sum (f_{\text{time}}(EP_1) + f_{\text{time}}(EP_2))/2 \\ &= (f_{\text{time}}(\text{critical path on } EP_1) \\ &\quad + f_{\text{time}}(\text{critical path on } EP_2))/2 \\ &= 34.5; \\ f_{\text{relia}}(REWG) &= \prod_{i=1}^{8} (\max\{f_{\text{relia}}(s_{i,j}) | j = 1, 2\}) \\ &= 0.85; \\ f_{\text{sec}}(REWG) &= \min\{(f_{\text{sec}}(s_{i,j}) | \\ &\quad j = 1, 2,\ i = 1, 2, \ldots, 8\} = 2. \end{aligned}$$

5.2 Computation of the Possible Worst QoS

On the contrary, the worst QoS means registry will compose services with the worst QoS metrics for each task, and inform the requester with the result. We define ***CLV*** as the worst QoS. ***CLV*** represents the possible worst composite service, which is the most economical. With the same meaning as that of ***CHV***,

CLV helps requester understand what the worst is, then specifies his required QoS.

Definition 3. ***CLV*** $= \langle clv_1, clv_2, \ldots, clv_k \rangle$, *where* $clv_i = f_{QoS\ metrics\ i}\ (REWG)$ *(with* $worst\{q^1_{j,i} | 1 \leqslant j \leqslant m\}, \ldots,\ worst\{q^2_{j,i} | 1 \leqslant j \leqslant m\}, \ldots,\ worst\{q^n_{j,i} | 1 \leqslant j \leqslant m\}$*), and worst = max (or min). The bigger (or smaller)* $q^x_{j,i}$ *is, the worse the QoS will be. For example, both higher price and smaller trust mean worse QoS.*

According to the calculation method of worst QoS for each metric in Section 4, the ***CLV*** of five common QoS metrics for Fig.1 with the assumed QoS in Table 1 is:

$$\begin{aligned}
f_{\text{price}}(REWG) &= \sum_{i=1}^{8}(\max\{f_{\text{price}}(s_{i,j}) | j = 1, 2\}) \\
&= 113; \\
f_{\text{trust}}(REWG) &= \sum_{i=1}^{8}(\min\{f_{\text{trust}}(s_{i,j}) | j = 1, 2\})/8 \\
&= 3.7; \\
f_{\text{time}}(REWG) &= \sum(f_{\text{time}}(EP_1) + f_{\text{time}}(EP_2))/2 \\
&= \sum(f_{\text{time}}(\text{critical path on } EP_1) \\
&\quad + f_{\text{time}}\ (\text{critical path on } EP_2))/2 \\
&= 38.5; \\
f_{\text{relia}}(REWG) &= \sum_{i=1}^{8}(\min\{f_{\text{relia}}(s_{i,j}) | j = 1, 2\}) \\
&= 0.6; \\
f_{\text{sec}}(REWG) &= \min\{(f_{\text{sec}}(s_{i,j}) | \\
&\quad j = 1, 2,\ i = 1, 2, \ldots, 8\} = 2.
\end{aligned}$$

5.3 Computation of the Average QoS

If it is not enough for the requester to have the knowledge of the best and the worst QoS, providing the average QoS to will help him with a deeper investigation about the QoS of all composite services. We define CAV as the average QoS.

The generation of CAV is much more complex, because it needs the QoS sum of all possible m^n composite services. Here, we propose to compute the average QoS reference vector for all possible composite services using the optimization computation method in Section 4 rather than listing down them all.

Definition 4. ***CAV*** $= \langle cav_1, cav_2, \ldots, cav_k \rangle$, *where in theory* $cav_j = \frac{\sum_{i=1}^{m^n} f_{QoS\ \ metrics\ \ j}(G_i)}{m^n}$ *and* G_i *is the i-th composite service. The following results are achieved according to the optimization calculation in Section 4.*

- *Price*

$$f_{\text{price}}(REWG) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m} = \frac{\sum_{i=1}^{2}\sum_{j=1}^{8} u_{i,j}}{2} = 96.$$

- *Trust*

$$f_{\text{trust}}(REWG) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}}{m \times n} \frac{\sum_{i=1}^{2}\sum_{j=1}^{8} u_{i,j}}{2 \times 8} = 4.3.$$

- *Security*

The process of computing ***CAV*** *element for security is given below. Table* 2 *lists the number of times that each QoS element appears in* $f_{\text{sec}}(REWG)$.

According to the definition of QoS matrix $\boldsymbol{U}_{\text{QoS metrics}}$, $\boldsymbol{U}_{\text{sec}} = \begin{bmatrix} 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 2 & 3 & 3 & 2 & 3 & 2 & 2 & 3 \end{bmatrix}$, Tag matrix

$$\boldsymbol{T} = \begin{bmatrix} u^{2,1}_{1,1} & u^{2,2}_{1,2} & u^{2,3}_{1,3} & u^{2,4}_{1,4} & u^{3,1}_{1,5} & u^{3,2}_{1,6} & u^{3,3}_{1,7} & u^{3,4}_{1,8} \\ u^{2,5}_{2,1} & u^{3,5}_{2,2} & u^{3,6}_{2,3} & u^{2,6}_{2,4} & u^{3,7}_{2,5} & u^{2,7}_{2,6} & u^{2,8}_{2,7} & u^{3,8}_{2,8} \end{bmatrix}.$$

Table 2. The number of Times that Elements Appearing in f_{sec} $(REWG)$

$u^*_{i,j}$	$x^*_{i,j}$
$3+7\varepsilon, 3+6\varepsilon, 3+5\varepsilon, 3+4\varepsilon,$ $3+3\varepsilon, 3+2\varepsilon, 3+\varepsilon, 2+5\varepsilon$	0
$2+4\varepsilon$	$1 \times 1 \times 1 \times 2 \times 2 \times 2 \times 2 = 16$
$2+3\varepsilon$	$1 \times 1 \times 1 \times 2 \times 2 \times 2 \times 2 = 16$
$2+2\varepsilon$	$1 \times 1 \times 2 \times 2 \times 2 \times 2 \times 2 = 32$
$2+\varepsilon$	$1 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$
2	$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 128$

After the update,

$$\boldsymbol{U}^*_{\text{sec}} = \begin{bmatrix} 2 & 2+\varepsilon & 2+2\varepsilon & 2+3\varepsilon & 3 & 3+\varepsilon & 3+2\varepsilon & 3+3\varepsilon \\ 2+4\varepsilon & 3+4\varepsilon & 3+5\varepsilon & 2+5\varepsilon & 3+6\varepsilon & 2+6\varepsilon & 2+7\varepsilon & 3+7\varepsilon \end{bmatrix}.$$

By using $\boldsymbol{U}^*_{\text{sec}}$, the computation steps and result of $x^*_{i,j}$ is given in Table 2. So we have:

$$\begin{aligned}
f_{\text{sec}}(REWG) &= ((2+4\varepsilon) \times 16 + (2+3\varepsilon) \times 16 + (2+2\varepsilon) \\
&\quad \times 32 + (2+\varepsilon) \times 64 + 2 \times 128)/256 \\
&= 2.
\end{aligned}$$

- Reliability

$$f_{\text{relia}}(REWG) = \frac{\prod_{j=1}^{n}\sum_{i=1}^{m} u_{i,j}}{m^n}$$

$$= \prod_{j=1}^{8} \Big(\sum_{i=1}^{2} u_{i,j} \Big) / 256 = 0.71.$$

• Time

Continuing with the example in Fig.1 and Table 1, the computing process for time of ***CAV*** is given as follows:

$$EP_1 = t_1 \sim (t_2 \wedge t_3) \sim t_4 \sim t_5 \sim t_6 \sim t_8.$$

The number of composite services supported by EP_1 is:

$$\lambda_1 = 2 \times 4 \times 2 \times 2 \times 2 \times 2 = 128.$$
$$EP_2 = t_1 \sim (t_2 \wedge t_3) \sim t_4 \sim t_5 \sim t_7 \sim t_8.$$

Similarly, for EP_2: $\lambda_2 = 2 \times 4 \times 2 \times 2 \times 2 \times 2 = 128$.

Average time of $EP_1 = (5+4)/2 + (9 \times 2 + 8 + 7)/4 + (5+4)/2 + (5+4)/2 + (5+4)/2 + (9+7)/2 = 34.2$.

Average time of $EP_2 = (5+4)/2 + (9 \times 2 + 8 + 7)/4 + (5+4)/2 + (5+4)/2 + (5+6)/2 + (9+7)/2 = 35.2$.

f_{time} $(REWG)$ = (average time of $EP_1 \times \lambda_1$ + average time of $EP_2 \times \lambda_2)/(\lambda_1 + \lambda_2) = 34.7$.

5.4 Generation of CRV

After registry informs the requester about ***CHV***, ***CLV*** and ***CAV***, the requester could determine his own QoS requirement according to these important information, and this QoS is defined as ***CRV*** which will be used for the target of service composition.

Definition 5. ***CRV*** $= \langle crv_1, crv_2, \ldots, crv_k \rangle$, *where* crv_i *is the QoS determined by the end user and will be used for reference to negotiate with the registry for service composition implementation in future.*

CRV should be produced based on above ***CHV***, ***CLV***, and ***CAV***. In the above example, the results of QoS metrics "$\langle$*price*, *time*, *trust*, *reliability*, *security*$\rangle$" are ***CHV***= $\langle 79, 34.5, 4.7, 0.85, 3 \rangle$, ***CLV***= $\langle 113, 38.5, 3.7, 0.6, 3 \rangle$, and ***CAV***= $\langle 96, 34.7, 4.3, 0.71, 3 \rangle$. The requester first selects one of them as his reference vector. This means to let ***CRV*** = ***CHV***, or ***CLV***, or ***CAV***. However, if he is not satisfied, he could continue to determine his really required QoS by assigning certain impact factor to ***CRV***. For example, some demands his required QoS to be better than ***CAV*** (e.g., 1.2 times of ***CAV***), other might need their QoS to be about 0.8 times of ***CHV***. To describe this kind of relationship, we introduce impact factor (*imf*) to influence the generation of ***CRV***, where $crv_i = imf \times crv_i$. The default value of *imf* is 1. Of course, the impact of *imf* cannot exceed the range of ***CLV*** and ***CHV***. This is assured by the following (1) strictly.

Besides the use of *imf*, the requester also could customize each metrics of ***CRV*** independently as long as the value locates between ***CLV*** and ***CHV***.

$$crv_i = \begin{cases} clv_i, & \text{if } imf \times crv_i < clv_i; \\ chv_i, & \text{if } imf \times crv_i > chv_i; \\ imf \times crv_i & \text{if } clv_i < imf \times crv_i < chv_i; \end{cases}$$
$$\text{if } clv_i < chv_i;$$
$$crv_i = \begin{cases} chv_i, & \text{if } imf \times crv_i < chv_i; \\ clv_i & \text{if } imf \times crv_i > clv_i; \\ imf \times crv_i, & \text{if } chv_i < imf \times crv_i < clv_i; \end{cases}$$
$$\text{if } clv_i \geqslant chv_i. \quad (1)$$

6 Selection of Implementation Algorithm for Composite Service

Given one workflow, assume there are m candidate services for each task, so there are m^n composite services which could be constructed, and selecting one from them can be simply fulfilled with exhaustive search when n is small. With the increase of n, selection algorithm is being confronted with a large number of combination choices. To solve this problem, some constraints and user preferences which restrict the composite service are introduced. With the restriction, service composition evolves into an NP-hard problem of constraint satisfaction[24]. Some papers consider it as complex models, such as multiple dimension Knapsack model or Multiple Choice Knapsack Problem[25], and multi-constraint optimal path problem[26]. The widely used linear integer programming model[10,27] and the classic genetic algorithm[14,28] are used to solve composition implementation. [11, 13] gives evaluation about some solutions, for example, genetic algorithms are more flexible than integer linear programming, but are less computationally efficient.

In this paper, exhaustive search and linear integer programming (IP)[29] used in [10, 11] are introduced to demonstrate the necessarity of algorithm selection because different algorithms bring about different composition results with different costs, exhaustive search can get the best result but it is feasible only in environment with small scale and IP is a popular solution in many research fields due to its better trade-off between result and cost. In fact, they represent two kinds of algorithms to execute accurate but slow selection and limited error permitted but fast selection.

6.1 Accurate Composition Algorithm

If a requester applies for the most accurate implementation of QoS requirement, that means the

composite service whose QoS is the closest to ***CRV*** is selected, the registry will recommend him the accurate selection algorithm (AS), at the same time, it also informs him about the long time to consume. Then the registry will form the QoS matrix of all composite services, do QoS normalization, and finally, select the one whose QoS has the lowest difference with ***CRV***.

The QoS matrix of all composite services is:

$$\boldsymbol{Q} = \begin{bmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,k} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ q_{m^n,1} & q_{m^n,2} & \cdots & q_{m^n,k} \end{bmatrix}.$$

6.1.1 QoS Matrix Normalization in Terms of CRV, CHV and CLV

Accurate service selection relies on service ranking. In order to rank all possible composite services, the QoS matrix $\boldsymbol{Q}$ needs to be normalized. This is important, because the interpretation of QoS semantic description of each metrics might be different. For some QoS metrics, the higher the value is, the higher the QoS will be; for others, the metrics understanding might be opposite. What we need is some uniform measurement mechanism independent of QoS property to normalize all metrics. Each element in matrix $\boldsymbol{Q}$ will be normalized using (2) below. It is based on ***CRV***, ***CHV*** and ***CLV***, and the normalized QoS matrix is $\boldsymbol{Q}'$.

$$\boldsymbol{Q}' = \begin{bmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,k} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ v_{m^n,1} & v_{m^n,2} & \cdots & v_{m^n,k} \end{bmatrix},$$

normalizing method:

$$v_{i,j} = \begin{cases} \left| \dfrac{q_{i,j} - crv_j}{chv_j - clv_j} \right|, & \text{if } chv_j - clv_j \neq 0; \\ 0, & \text{if } chv_j - clv_j = 0. \end{cases} \quad (2)$$

Theorem 2. *For any element $v_{i,j}$ in matrix $\boldsymbol{Q}'$, there exists $v_{i,j} \in [0,1]$.*

Proof. For $\forall q_{i,j}$, if $clv_j \leqslant chv_j$, there is:

$$clv_j \leqslant q_{i,j} \leqslant chv_j. \quad (3)$$

According to (1), $clv_j \leqslant crv_j \leqslant chv_j$. So

$$-chv_j \leqslant -crv_j \leqslant -clv_j. \quad (4)$$

Combining (3) and (4), the result is:

$$clv_j - chv_j \leqslant q_{i,j} - crv_j \leqslant chv_j - clv_j.$$

Hence, there is $-1 \leqslant \frac{q_{i,j}-crv_j}{chv_j-clv_j} \leqslant 1$, which means

$$v_{i,j} \in [0,1].$$

Because of the same reason, for $\forall q_{i,j}$, if $clv_j > chv_j$, there is $-1 \leqslant \frac{q_{i,j}-crv_j}{clv_j-chv_j} \leqslant 1$, that also means $v_{i,j} \in [0,1]$.

Thus, Theorem 2 is proved. □

From (2), it is clear that the closer to ***CRV*** the $q_{i,j}$ is, the lower the result $v_{i,j}$ will be. Therefore, the lowest value of $v_{i,j}$ should be selected preferentially.

6.1.2 Accurate Service Selection

Depending on $\boldsymbol{Q}'$, accurate selection (AS) algorithm through exhaust search can always produce the most optimal solution. Furthermore, any flexible priority-based selection policy can also be implemented. As an example, the weight policy version is presented here. Weight policy is a general decision-making policy used in many situations, and the requester could express his preferences about QoS metrics by providing weight. The weight vector is:

$$\boldsymbol{WV} = \langle wv_1, wv_2, \ldots, wv_k \rangle$$

where $wv_i \in [0,1]$ and $\sum_{i=1}^{k} wv_i = 1$. The higher the value is, the more preferable the user will feel. ***WV*** works only after QoS matrix has been normalized. The following formula can be used to compute the overall quality score for each composite service:

$$\text{Score}(G_i) = \sum_{j=1}^{k} v_{i,j} \times wv_j.$$

And the registry will choose one with the lowest score from the m^n possible composite services, that is $\min\{\text{Score}(G_i) | i = 1, 2, \ldots, m^n\}$. Although exhaust search can achieve accurate service selection, the drawback is the high overheads. It constructs all possible composite services and compares their QoS with the time and memory complexity of $O(m^n)$. Hence, it is only suitable when the numbers of tasks, candidate services, and QoS metrics are small. When n is getting bigger we propose the following integer programming approach because the problem is known to be NP-hard.

6.2 Fast Composition Algorithm

In this subsection, a fast composition algorithm based on 0-1 IP is proposed to select a good composite service without constructing all the possible composite services. There are three inputs in an IP problem: a set of variables, an object function, and a set

of constraints, where both the objective function and the constraints must be linear. IP attempts to maximize or minimize the value of the objective function by adjusting the values of the variables while enforcing the constraints. The output of IP problem is the maximum or minimum value of the objective function and the values of variables to get the maximum or minimum value. The problem of selecting an optimal composition is mapped into IP problem as follows:

First, for each service $s_{i,j}$, an integer variable $x_{i,j}$ is used to present whether $s_{i,j}$ is selected: $x_{i,j}$ is 1 if service $s_{i,j}$ is selected, and 0, otherwise.

Second, referring back to our QoS metrics, we see that since the computation of reliability is the product of all individual reliability, it is nonlinear and cannot be computed by Integer Programming. As a result, we propose to transform it into linear function by applying algorithm *ln* as follows.

$$f(REWG) = \langle f_{\text{price}}(REWG), f_{\text{time}}(REWG), f_{\text{relia}}(REWG), f_{\text{trust}}(REWG), f_{\text{sec}}(REWG)\rangle,$$

where

$$f_{\text{price}}(REWG) = \sum_{j=1}^{m}\sum_{i=1}^{n} x_{i,j} f_{\text{price}}(s_{i,j});$$

$$f_{\text{time}}(REWG) = \sum_{k=1}^{a}\Big(\sum_{j=1}^{m}\sum_{i=1}^{n} x_{i,j} \times f_{\text{time}}(s_{i,j}|s_{i,j} \text{ is on the critical path of } GEP_k)\Big)\Big/\alpha$$

$$f_{\text{trust}}(REWG) = \sum_{j=1}^{m}\sum_{i=1}^{n} x_{i,j} f_{\text{trust}}(s_{i,j})/n;$$

$$f_{\text{relia}}(REWG) = \ln\Big(\prod_{j=1,i=1}^{i=m,j=n}(x_{i,j} f_{\text{relia}}(s_{i,j})|x_{i,j}=1)\Big) = \sum_{j=1}^{m}\sum_{i=1}^{n} x_{i,j}\ln(f_{\text{relia}}(s_{i,j}));$$

$$f_{\text{sec}}(REWG) = \max\{x_{i,j} \times f_{\text{sec}}(s_{i,j})| \quad i=1,2,\ldots,n, \quad j=1,2,\ldots,m\}.$$

Therefore, the objective here is to find a set of services, such that $Obj = h(f(REWG), \boldsymbol{CRV})$ is minimized.

$$\begin{aligned} Obj =& h(f(REWG), \boldsymbol{CRV}) \\ =& \frac{|f_{\text{price}}(REWG) - crv_{\text{price}}|}{chv_{\text{price}} - clv_{\text{price}}} \\ &+ \frac{|f_{\text{time}}(REWG) - crv_{\text{time}}|}{chv_{\text{time}} - clv_{\text{time}}} \\ &+ \frac{|f_{\text{trust}}(REWG) - crv_{\text{trust}}|}{chv_{\text{trust}} - clv_{\text{trust}}} \\ &+ \frac{|f_{\text{relia}}(REWG) - crv_{\text{relia}}|}{chv_{\text{relia}} - clv_{\text{relia}}} \\ &+ \frac{|f_{\text{sec}}(REWG) - crv_{\text{sec}}|}{chv_{\text{sec}} - clv_{\text{sec}}}. \end{aligned} \tag{5}$$

Finally, the constraints are described in two parts. The first part of constraint is:

$$\begin{cases} \sum_{i=1}^{n} x_{i,j} = 1; \\ \sum_{j=1}^{m}\sum_{i=1}^{n} x_{i,j} = n. \end{cases} \tag{6}$$

Besides the selection constraint, each element should not deviate from the $\boldsymbol{CRV}$ too much. In other words, another constraint of error variable vector needs to be defined as $\boldsymbol{\delta} = \langle \delta_{\text{price}}, \delta_{\text{time}}, \delta_{\text{relia}}, \delta_{\text{trust}}, \delta_{\text{sec}}\rangle$. It is noted that some error variables can be set to zero if necessary. The second part of constraint inequalities is:

$$\begin{cases} |f_{\text{price}}(REWG) - crv_{\text{price}}| \leqslant \delta_{\text{price}}; \\ |f_{\text{time}}(REWG) - crv_{\text{time}}| \leqslant \delta_{\text{time}}; \\ |f_{\text{relia}}(REWG) - crv_{\text{relia}}| \leqslant \delta_{\text{relia}}; \\ |f_{\text{trust}}(REWG) - crv_{\text{trust}}| \leqslant \delta_{\text{trust}}; \\ |f_{\text{sec}}(REWG) - crv_{\text{sec}}| \leqslant \delta_{\text{sec}}. \end{cases} \tag{7}$$

7 Experimental Confirmation

Exhaustive search will consume much longer time with more accurate results than integer programming — everybody would expect that. In order to confirm this conclusion and demonstrate the necessity for users to choose suitable composition algorithms by themselves, we conducted experiments to compare how the numbers of QoS metrics, tasks, candidate services affect the selection cost and compare their service composition results.

The first four elements in (5) are linear and they could be expressed as 0-1 in the Integer Programming. However, the last element cannot be done in this way, so the algorithm we adopt is mixed IP. In our simulation, we still could get the approximate solution, not the total minimization of sum of five elements. We first remove the absolute sign in (5) by dividing the solution space into 16 sub-spaces, then the 16 sub-minimums associated with the first four elements are obtained. In this way, 16 composite services are found. Next, considering the security QoS by adding $\frac{|f_{\text{sec}}(REWG) - crv_{\text{sec}}|}{chv_{\text{sec}} - clv_{\text{sec}}}$ to the 16 sub-minimums, and selecting the one with the smallest value, we can obtain the result. Hence, our final objective is the minimum of 16 sub-minimums. This linear minimization can be calculated using MILP

function of Matlog, which is used for solving 0-1 integer planning in Matlab. All QoS values are generated randomly using Matlab.

7.1 Comparison of Composition Cost

In our first set of simulation, we study the effect of three different parameters on the selection cost. The service selection cost here is defined as the duration used to select services.

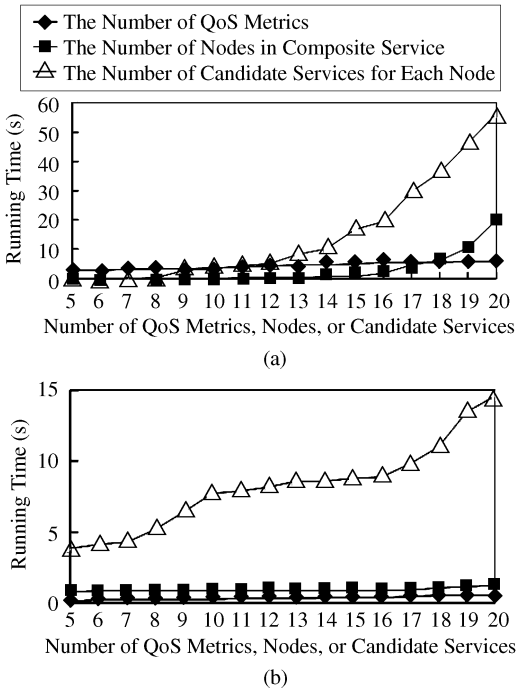


Fig.4. Impact of different parameters on execution cost. (a) For accurate selection. (b) For fast selection.

We change one parameter while fixing the other two to find out the parameter impact. Fig.4 shows the result, where the x-axis represents the parameter and y-axis means the corresponding selection cost. It is clearly shown that the service selection cost increases when each of the three parameters increases. The main difference is that the impact of the number of candidate services is much larger than that of the other two.

From Fig.4, we also see that the computation of accurate search is much higher than that of the fast selection. For example, when the number of candidate services is 16, the consumed selection time is less than 9s for fast selection but more than 20s for accurate selection.

7.2 Comparison of Composition Result

The second set of experiments aims at evaluating the QoS of composite service with a given ***CRV***. AS will find the most appropriate composite service whose QoS is the closest to ***CRV*** while IP will find a solution under the given constraints. The dimension of QoS vector studied here is 5. We also assume that there are 10 candidate services for each task. The QoS values of all candidates are randomly generated. The experiment is repeated six times, the element values of ***CRV*** are generated each time, and $\boldsymbol{\delta}$ is also set each time.

The result is shown in Table 3. The last row lists the ***CRV*** for the 6th result. The columns with label AS represent each element of composite QoS which is achieved by accurate selection, and the columns with label IP represent each element of composite QoS which is achieved by fast selection. The last two columns are the values of $Obj = h(f(REWG), \boldsymbol{CRV})$, which is computed by (5) according to the distances between ***CRV*** and the composite QoS vector of AS and IP respectively.

Table 3 shows that AS gives better QoS of composite service than IP. The distance between QoS via AS and the ***CRV*** is less than that via IP for the given ***CRV***. This is expected, because AS performs exhaustive search in the entire solution space. On the contrary, the IP method only aims at achieving an approximate solution with each element in composite QoS to be constrained to a range. It still could be inferred that the distances of AS and IP are equal to each other when the solution corresponding to the global minimum is fitted

Table 3. Comparison of Composite QoS

G_i	f_{price}		f_{time}		f_{trust}		f_{relia}		f_{sec}		Obj	
	AS	IP	AS	IP	AS	IP	AS	IP	AS	IP	IP	AS
1	159	206	0.4	0.7	4.4	2.1	0.2	0.3	4.0	5.0	1.38	0.70
2	269	207	0.8	0.8	4.2	2.2	0.2	0.3	1.0	5.0	1.45	0.67
3	136	179	0.9	0.7	2.6	1.6	0.1	0.3	1.0	5.0	1.44	0.54
4	140	200	0.8	0.8	4.2	1.9	0.2	0.4	2.0	5.0	1.46	0.59
5	150	208	0.6	0.8	3.7	1.8	0.3	0.4	1.0	5.0	1.38	0.57
6	59.7	136	0.6	0.8	2.4	1.9	0.3	0.2	3.0	5.0	1.43	0.56
CRV	68.7	68.7	0.34	0.34	5.00	5.00	0.39	0.39	1.00	1.00		

into constrains of the IP method. Hence, for single QoS metric, the results of AS and IP will be very close to each other.

Performing exhaustive search is NP hard and the cost is high, especially when the number of candidate service increases. Though IP policy might not give the optimal answer, it does good trade-off between the composition cost and composite QoS, as is shown in the table.

The error variable vector $\boldsymbol{\delta} = \langle \delta_{\text{price}}, \delta_{\text{time}}, \delta_{\text{relia}}, \delta_{\text{trust}}, \delta_{\text{sec}} \rangle$ also affects the result. For example, the global minimization of *Obj* is assured when elements in $\boldsymbol{\delta}$ increase. And no solution might be found if they are too small (because no result satisfying (5), (6) and (7)). Under this situation, requester and registry can keep negotiating to tune $\boldsymbol{\delta}$ higher until one solution is found.

8 Conclusion and Future Work

In this paper, we propose to strengthen current service composition mechanism by adding QoS requirement generation and its composition implementation algorithm selection mechanism based on the QoS reference vectors which are generated to represent QoS overview of all possible composite services with the given workflow. The computation methods of ***CHV***, ***CLV***, ***CAV*** are presented and studied in detail. The mechanism is described and embedded in the improved SOA framework. Two selection algorithms which represent two kinds of algorithms are introduced to execute experiment and compared. The accurate selection leads to better QoS with higher execution duration, which is more consistent with the requester required SLA. Just the opposite, the fast selection composes a service satisfying QoS constraints with trade-off between result and cost.

Note that the approach proposed in this paper is to construct composite service before it runs, we would like to keep the stabilization of composite service, so even if some QoS problems happen, frequent, and dynamic component service switch should be avoided. Therefore, our focus will turn to enhancing QoS for runtime composite service for future research. This is important, because composite service published on the Internet will often confront with many QoS challenges dynamically. This includes non-deterministic overload such as Hot-Spot and DDoS attack.

References

[1] Curbera F *et al.* Unraveling the web services: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 2002, 6(2): 86–93.

[2] Curbera F, Goland Y, Klein J, Leymann F, Roller D, Thatte S, Weerawarana S. Business process execution language for web services. Version 1.1, 2002, http://www.ibm.com/developerworks/library/specification/ws-bpel.

[3] Patil S, Newcomer E. ebXML and Web services. *IEEE Internet Computing*, May/June 2003, 7(3): 74–82.

[4] Gorbenko A, Kharchenko V, Romanovsky A. On composing dependable web services using undependable web components. *International Journal of Simulation and Process Modelling*, 2007, 3(1/2): 45–54.

[5] Claro D B, Macêdo R J A. Dependable web service compositions using a semantic replication scheme. In *Proc. Anais of the VI Brazilian Symposium of Networks and Distributed Systems*, Rio de Janeiro, Brazil, May 2008.

[6] Cardoso J, Sheth A P, Miller J A, Arnold J, Kochut K J. Modeling quality of service for workflows and web service processes. *Web Semantics Journal: Science, Services and Agents on the World Wide Web Journal*, 2004, 1(3): 281–308.

[7] Yu J, Han Y B, Han J *et al.* Synthesizing service composition models on the basis of temporal business rules. *Journal of Computer Science and Technology*, 2008, 23(6): 885–894.

[8] Xu D, Nahrstedt K. Finding service paths in a media service proxy network. In *Proc. the SPIE/ACM Multimedia Computing and Networking Conference* (*MMCN*), San Jose, USA, 2002, pp.171–185.

[9] Patel C, Supekar K, Lee Y. A QoS oriented framework for adaptive management. In *Proc. the Conference of Web Service Based Workflows Database and Expert Systems* (*DEXA-2003*), Prague, Czech Republic, 2003, pp.826–835.

[10] Zeng L, Benatallah B, Ngu A H H, Dumas M, Kalagnanam J, Chang H. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, May 2004, 30(5): 311–327.

[11] Ardagna D, Pernici B. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 2007, 33(6): 369–384.

[12] Bonatti P A, Festa P. On optimal service selection. In *Proc. the 14th International Conference on World Wide Web*, Chiba, Japan, 2005, pp.530–538.

[13] Jaeger M C, Muhl G, Golze S. QoS-aware composition of web services: An evaluation of selection algorithms. In *Proc. International Conference on Cooperative Information Systems*, Agia Napa, Cyprus, 2005, pp.646–661.

[14] Canfora G, Penta M, Esposito R, Villani M L. QoS-aware replanning of composite web services. In *Proc. International Conference on Web Services* (*ICWS'05*), Orlando, USA, 2005, pp.121–129.

[15] Wohlstadter E, Tai S, Mikalsen T A, Rouvellou I, Devanbu P T. GlueQoS: Middleware to sweeten quality-of-service policy interactions. In *Proc. the International Conference on Software Engineering* (*ICSE'04*), Edinburgh, England, 2004, pp.189–199.

[16] Chhetri M B, Lin J, Goh S K, Yan J, Zhang J Y, Kowalczyk R. A coordinated architecture for the agent-based service level agreement negotiation of web service composition. In *Proc. the Australian Software Engineering*, Sydney, Australia, 2006, pp.90–99.

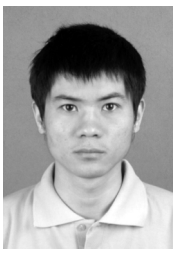
[17] Jing-Fan Tang, Xiao-Liang Xu. An adaptive model of service composition based on policy driven and multi-agent negotiation. In *Proc. International Conference on Machine Learning and Cybernetics*, Dalian, China, 2006, pp.113–118.

[18] Cao J, Wang J, Zhang S, Li M. A multi-agent negotiation based service composition method for on-demand service. In *Proc. IEEE International Conference on Services Computing*, Orlando, USA, 2005, pp.329–332.

[19] http://rubis.objectweb.org/.
[20] Menasce D A. QoS issues in web services. *IEEE Internet Computing*, 2002, 6(6): 72–75.
[21] Liu Y, Ngu A H, Zeng L Z. QoS computation and policing in dynamic web service selection. In *Proc. the World Wide Web Conference*, New York, USA, 2004, pp.66–73.
[22] European Commission. IT security evaluation criteria. V. 1.2, Office for Official Publications of the EC, 1991, http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf.
[23] Common criteria for IT security evaluation. V. 3.1, Common Criteria Implementation Board. 2006, http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R1.pdf.
[24] Aggarwal R, Kunal Verma, Miller J, Milnor W. Constraint driven web service composition in METEOR-S. In *Proc. 2004 IEEE International Conference on Services Computing* (*SCC 2004*), Shanghai, China, 2004, pp.23–30.
[25] Yu T, Lin K J. Service selection algorithms for Web services with end-to-end QoS constraints. *Journal of Information Systems and E-Business Management*, 2005, 3(2): 103–126.
[26] Yu T, Lin K J. Service selection algorithms for composing complex services with multiple QoS constraints. In *Proc. Service-Oriented Computing* (*ICSOC 2005*), *Lecture Notes in Computer Science*, Amsterdam, Netherlands, pp.130–143.
[27] Ardagna D, Pernici B. Global and local QoS guarantee in web service selection. In *Proc. the Business Process Management Workshop* (*BPM'05*), Nancy, France, 2005, pp.32–46.
[28] Claro D B, Albers P, Hao J K. Selecting web services for optimal composition. In *Proc. International Workshop on Semantic and Dynamic Web Processes* (*ICWS*), Orlando, USA, Jan. 10–15, 2005.
[29] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. Second Edition, MIT Press, 2001.

Bang-Yu Wu received her M.S. degree and worked in Jiangnan Institute of Computing Technology. She is currently a Ph.D. candidate in the Department of Computer Science and Technology and the School of Software, Tsinghua University. Her main research areas are parallel and distributed system, information security and software service engineering.

Chi-Hung Chi received his Ph.D. degree from Purdue University. Before he joined Tsinghua University in 2005, he worked for Philips Laboratories, IBM Poughkeepsie, Chinese University of Hong Kong and National University of Singapore, respectively. His main research areas are content networking, systems and network security, and software service engineering.



Shi-Jie Xu received is M.SE. degree from the School of Software, Tsinghua University in July 2007. His main research area is software service engineering.

Ming Gu is a professor at the School of Software, Tsinghua University, China. Her research interests include software system theory, middleware, information security, and enterprise information systems. She is concurrently the vice-executive dean of the School of Software, Tsinghua University.

Jia-Guang Sun is a professor at the School of Software, Tsinghua University, China. Prof. Sun, widely regarded as a leader in CAD research in China, is a member of the Chinese Academy of Engineering. Prof. Sun is the director of the National Engineering Research Center for CAD Supporting Software and is concurrently the executive dean of the School of Software, Tsinghua University and of vice-chairman of Committed of the National Natural Science Foundation of China.