ISeB (2005) 3: 103–126
DOI 10.1007/s10257-005-0052-z

ORIGINAL ARTICLE

Tao Yu · Kwei-Jay Lin

Service selection algorithms for Web services with end-to-end QoS constraints

Abstract Web services are new forms of Internet software that can be universally deployed and invoked using standard protocols. Services from different providers can be integrated into a composite service regardless of their locations, platforms, and/or execution speeds to implement complex business processes and transactions. In this paper, we study the end-to-end QoS issues of composite services by utilizing a QoS broker that is responsible for selecting and coordinating the individual service component. We design the service selection algorithms used by QoS brokers to construct the optimal composite service. The objective of the algorithms is to maximize the user-defined utility function value while meeting the end-to-end delay constraint. We propose two solution approaches to the service selection problem: the combinatorial approach, by modeling the problem as the Multiple Choice Knapsack Problem (MCKP), and the graph approach, by modeling the problem as the constrained shortest path problem in the graph theory. We study efficient solutions for each approach.

Key words Web service · QoS broker · Service compostion · Service selection · End-to-end constraint

1 Introduction

The Web services framework has evolved as the software foundation for next generation enterprise and Web-based systems. By adopting standard-based protocols SOAP (Gudgin et al. 2003), WSDL (Christensen et al. 2001) and

This research was supported in part by NSF CCR-9901697.

T. Yu · K.-J. Lin (✉)
Dept. of Electrical Engineering and Computer Science, University of California, Irvine, Irvine, California 92697-2625, USA. E-mail: Klin@uci.edu

UDDI (OASIS Universal), service components from different service providers can be conveniently integrated into a *composite service* regardless of their locations, platforms and/or execution speeds. Instances of Web services may now interact with each other, fulfilling individual requests that carry out parts of complex transactions or workflows. As more composite Web services are deployed, many of them may share common services such as real-time data services, search engines, supply chains, etc. With growing demands, many service providers have started to offer different QoS service levels (SLA's) so as to meet the needs of different user groups, by offering different service qualities based on user needs or service cost. One example of such projects is the Oceano project at IBM (The Oceano and IBM).

For composite services, one of the QoS issues is to define the service integration model and identify the optimal service selection to meet a user's QoS requirement. Due to the dynamic nature of Web services, there are several basic system properties that must be considered:

- The set of candidate services that are capable of providing a given functionality may be large and constantly changing;
- The quality of a composite service is measured by its end-to-end quality, rather than any individual service component only. Moreover, the end-to-end quality is decided collectively by all individual service components;
- For a composite Web service, there may be many possible ways to fulfill a business process;
- The services that make up a business process may continue to evolve. A business process also needs to accommodate to the system variations on a global level, such as service failure and upgrade;

Due to these considerations, a rigorous mechanism is needed to ensure the end-to-end quality of a composite Web service. Our research on the QoS of composite services has proposed the QoS-Capable Web Service architecture, QCWS (Chen et al. 2003), that implements a QoS broker between Web service clients and providers. In QCWS, every service provider is assumed to offer many service levels for a service functionality. A QoS broker collects the QoS information of candidate service providers (servers), makes service (and service level) selection decisions for clients, and then checks with servers to receive QoS service commitments. Although a broker is modeled as a separate entity from clients and servers, a QoS broker may be practically implemented as part of a client, part of a server, or an independent Web service. The functionalities of a broker may vary slightly depending on the setting of the broker.

In this paper, we study the service selection algorithms used by QoS brokers. A broker receives service requests from clients and identifies services that may meet the functional and QoS needs of those requests. The service selection algorithm considers service cost, service response time, server load, and network delay to make the best selection decision, under the end-to-end delay constraint. We have studied several selection algorithms and compared their performance using randomly generated test cases. Our study shows that the problem can be efficiently solved. The proposed algorithms may be adopted by QoS brokers to make dynamic on-line decisions.

The contribution of this research is as follows:

1. We define the QoS broker service mechanism for managing *end-to-end* QoS for composite Web services. The QoS broker service can work with existing Web service standards, such as the process service in the BPEL process composition, the coordinator service in WS-Transaction, and the registry service for UDDI registries. The QoS broker is designed to make service selection for client requests, based on their QoS constraints and requirements.
2. The objective function (called *utilities* in this paper) and constraint used by the proposed algorithms are based on a comprehensive set of system parameters, including static server information (service level), client QoS requirement (QoS constraint), dynamic server capacity (service benefit), and network communication delay. They could also consider service reliability, availability, etc. Moreover, the definition of a composite service includes individual services that may be structured as a sequence or a DAG (Directed Acrylic Graph). Each individual service may itself be a composite service or a single service.
3. We study the end-to-end QoS service selection using two different approaches, the combinatorial approach and the graph approach, and compare the pros and cons of the two approaches.
4. We present several service selection algorithms in both combinatorial and graph approaches. The most suitable algorithm depends on the structure of the composite service, the number of candidates for each service group, and the number of constraint definitions. These algorithms can be used by a QoS broker to select the most suitable services and service levels for clients and to construct optimal business processes.

The rest of this paper is organized as follows. Section 2 presents some related work. Section 3 provides the QoS models for Web service composition. Section 4 presents the system model defined for service selection algorithms. Several algorithms in both combinatorial and graph approaches for the service selection problem are shown in Sect. 5, with their performance compared. The paper is concluded in Sect. 6.

2 Related work

The end-to-end Web services selection is an important part of the Web service composition problem. Many have worked on this topic. Industrial standard specifications have been proposed. One of the recent proposals is BPEL4WS (Business Process Execution Language for Web services) (Curbera et al. 2003) which combines Microsoft's XLANG (Thatte 2001) and IBM's WSFL (Web service Flow Language) (Leymann 2001). It is positioned to be the standard for Web service composition. It provides a language for the formal specification of business processes and business interaction protocols. BPEL4WS can model the behavior of both executable and abstract processes. Other proposals include WSCI (Web service Choreography Interface) (Arkin et al. 2002), an XML-based interface description

language that describes the flow of messages exchanged by Web services in choreographed interactions, and BPML (Business Process Modeling Language) (BPMI org 2002), which is intended for expressing abstract and executable processes that address all aspects of enterprise business processes.

In contrast to these industrial standards, researchers in academics are developing an ontology of services, called DAML-S (DARPA Agent Markup Language) (Ankolekar et al. 2001). DAML-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in an unambiguous, computer-interpretable form. DAML-S markup of Web services will enable automated Web service discovery, invocation, inter-operation, composition and execution monitoring.

Many projects have studied Web service composition. The SWORD project (Ponnekanti and Fox 2002) provides a simple and efficient mechanism for Web service composition. It uses a rule-based expert system to check whether a composite service can be realized by existing services and to generate the execution plan. SWORD performs the planning at the composition time, not at run time. The advantage of offline planning is predictability and efficiency since no overhead is incurred at run time. However, the dynamic nature of Web services may cause an offline plan useless or inefficient since a pre-selected Web service may no longer be available or some new powerful services may become available.

In VanderMeer et al. (2003), authors describe a framework, called FUSION, for dynamic Web service composition and automatic execution. They stress on automatically generating an optimal execution plan from the abstract requirements that a user may specify, executing according to the plan, verifying the result against a user's stated satisfaction criteria and, finally, initiating the appropriate recovery procedures in case of satisfaction failure. The satisfaction criteria are defined by users, which are used to measure whether the actual result is acceptable or not.

Neither SWORD nor FUSION addresses QoS issues such as service latency, availability, reliability, that are critical to dynamic Web service composition and determine the performance of the whole system.

eFlow system (Casati et al. 2000) supports specification, enactment, and management of composite e-services, which are modeled as processes that are enacted by a service process engine. In eFlow, each node has service selection rules that can be based on some value-added information (Such as QoS). When an eFlow engine tries to execute an activity it calls a service broker that executes the service selection rule and returns a list of choices. Service selection rules are defined using a service broker-specific language, which include arbitrary service selection policies.

Patel, Supekar and Lee (Patel et al. 2003) propose a QoS-oriented framework WebQ for adaptive management of Web service based workflows. WebQ conducts the adaptive selection process and simultaneously provides binding and execution of Web services for the underlying workflow. Their proposed QoS selection criterion only considers service load and makes only local decisions.

All of the above systems have a centralized operator/engine/coordinator to control the execution of constructed composite services. That is, their

executions are all centralized. The centralized execution suffers some problem such as bottleneck and scalability. There are also systems in which the constructed composite services are executed in a decentralized/peer-to-peer manner. Such as SELF-SERV platform presented in Sheng et al. (2002). SELF-SERV relies on a declarative language for composing services based on state-charts. Web services are grouped into different service communities according to the functionalities they provided. They are declaratively composed, and the resulting composite services are executed in a peer-to-peer and dynamic environment among service communities. There is no centralized coordinator to control the service execution, so it does not suffer of the scalability and availability problem. Same as SWORD and FUSION systems, no QoS issues have been addressed in SELF-SERV.

The QoS problem in Web service composition is an end-to-end QoS problem. None of the previously mentioned projects consider the end-to-end QoS constraint/requirements. Zeng et al. (2003) gives a quality driven approach to select component services during execution of a composite service. They consider multiple QoS criteria such as price, duration, reliability and take into account of global constraints. The linear programming method is used to solve the service selection problem, which is usually too complex for run-time decisions.

Shankar et al. (1999) give an end-to-end QoS model and management architecture for complex distributed real-time system. In their model, a system contains several dependent components (tasks), the outputs of some tasks are the inputs of others and the output quality of every task depend on the input quality. QoS requirements are end-to-end in the user's point of view. They focus on how to do the service establishment in this architecture and give the QoS negotiation and establishment protocol.

3 QoS models for Web service composition

Future business systems require a seamless integration of many business processes, business applications, business intelligence, and Web services over the Internet. Delivering quality services to meet user needs is a significant and critical challenge because of the dynamic and unpredictable nature of business applications and Internet traffic. Business applications with very different characteristics and requirements compete for resources used to provide Web services. Without a careful management of service quality, critical business applications may suffer detrimental performance degradation, and result in functional failures and/or financial losses.

3.1 Web service QoS attributes

The area of QoS management covers a wide range of issues to match the needs of service requesters with those of the service providers. QoS has been a major area of interest in communication networks, real-time computing,

and multimedia systems. For Web services, QoS guarantee and enhancement have started to receive great attention.

In our current study, we consider four quality attributes as part of the Web service parameters (shown in Table 1). These QoS attributes can also be applied to evaluate QoS of the constructed business process. Since our goal is to build automated brokers for Web services, the QoS attributes that we consider must be intuitive to understand and easy to measure. These attribute data can be collected automatically without any user intervention.

Users may have constraints on one or more QoS attributes in their QoS requirements. In this paper, we only consider the case with only one QoS constraint, such as response time. The study for composition flows with two or more QoS constraints will be pursued in our future work.

3.2 Composition flow models

The QoS attributes of a business process are decided by the QoS attributes of individual services and their composition relationships. There are different ways that individual services can be integrated to form a business process. The four basic relationships are: (1) sequential; (2) parallel; (3) conditional; (4) loop. Figure 1 shows these four basic models. In this paper, we only consider the sequential composition model, which is the fundamental one. All the other models can be converted into sequential model. We can show how to do the conversions in our future report.

As shown in Table 1, the service response time includes service time and transmission time. In a composite service, the transmission time is the time needed to send a user's request to the first server, to pass the result of one server to the next server in the sequential model, and to return results to the user from the last server in the server chain. The transmission time estimation is different in combinatorial approach and graph approach. Figure 1(1) shows a business process that is composed by individual services sequentially. Each individual service and the business process have QoS attributes of response time, cost, availability and reliability. Table 2 shows the aggregation functions for compute the QoS attributes of the business process shown in Figure 1(1).

1. Response time: The response time of the business process s is the sum each individual service s_i's response time at the chosen service level.
2. Cost: The cost of the business process s is the sum of each individual service s_i's cost at the chosen service level.
3. Reliability: The reliability of the business process s is the product of each individual service s_i's reliability at the selected service level. It is a non -linear function. In order to make all aggregation functions to be linear, we can transform it using an logarithmic function. Suppose the reliability of the composite service is $R_s = \prod_{i=1}^{n} R_{s_i}$. By applying logarithmic function log, we obtain $log(R_s) = log(\prod_{i=1}^{n} R_{s_i}) = \sum_{i=1}^{n} log(R_{s_i})$. Let $R_s^{'} = log(R_s)$, $R'_{s_i} = log(R_{s_i})$, we have a linear function $R'_s = \sum_{i=1}^{n} R'_{s_i}$
4. Availability: The availability of the business process s is the product of each individual service s_i's availability at the selected service level. Same as the reliability attribute, we can use an logarithmic function to convert

Table 1 QoS Parameters

QoS parameters	Description	Value
Response time (T)	The time interval between when a user requests the service and when the user receives the response. It includes service time and transmission time. *Service time* : time to process a service request; *Transmission time*: time to send a request to server and get a result from the server (round-trip delay);	Service time is specified by service provider SLA. Transmission time is decided by the network load. $T = T_{sr} + T_{tr}$ T_{sr}: service time; T_{tr} : transmission time.
Cost (C)	It includes service cost and transmission cost. *Servicecost* : the cost for executing the service; *Transmission cost* : the cost for transmitting result data from a server to a requester;	Service cost is specified by service provider SLA. Transmission cost is decided by network operator. $C = C_{sr} + C_{tr}$ C_{sr}: service cost; C_{tr}: transmission cost.
Service availability (A)	The probability that a service is available.	It is computed from historical data. $A = T_a / T_t$ T_a: Amount of time that service is available; T_t: Total time monitored.
Service reliability (R)	The probability that a request is correctly handled within the expected time.	It is computed from historical data. $R = N_s / N$ N_s: Number of requests successfully responded; N: Total requests.

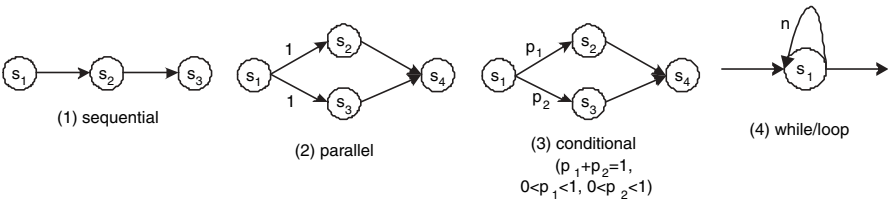

Fig. 1 Composition flow models

Table 2 QoS parameter aggregation for sequential composition

Attribute	Aggregate function
Response time	$T_s = \sum_{i=1}^{n} T_{s_i}$
Cost	$C_s = \sum_{i=1}^{n} C_{s_i}$
Reliability	$R_s = \prod_{i=1}^{n} R_{s_i}$
Availability	$A_s = \prod_{i=1}^{n} A_{s_i}$

it to a linear formulation. Let $A_s^{'} = log(A_s)$, $A'_{s_i} = log(A_{s_i})$, we thus have $A'_s = \sum_{i=1}^{n} A'_{s_i}$.

To construct an optimal composite Web service (business process), in addition to meeting user's functional and QoS requirements, we could define an *objective function* for optimization. In our study, we define a utility function as the optimization objective. The utility function is based on a rich set of system parameters, including static server information (service level), client QoS requirement (QoS constraint), dynamic server capacity (server load), and etc. Each service level of every individual service has its own utility computed by the utility function. Detailed information about utility function will be discussed in the next section. Suppose the utility of the chosen service level in each individual service s_i is U_{s_i}. The aggregate function of the utility of the sequential composite service is $U_s = \sum_{i=1}^{n} U_{s_i}$.

All QoS attributes and utility's aggregate functions are linear functions, as shown in Table 3. The assumption must be true for our service selection algorithms to work correctly.

Table 3 Linear aggregate function for sequential composition

Attribute	Aggregate function
Response time	$T_s = \sum_{i=1}^{n} T_{s_i}$
Cost	$C_s = \sum_{i=1}^{n} C_{s_i}$
Reliability	$R'_s = \sum_{i=1}^{n} R'_{s_i}$
Availability	$A'_s = \sum_{i=1}^{n} A'_{s_i}$

4 System model

In this section, we present the system model of our proposed service selection algorithm.

4.1 Notations and definitions

The notations used in this paper are defined as follows:

1. S is a service class. A *service class* is a collection of individual Web services with a common functionality but different non-functional properties (e.g. different locations, different quality levels, etc.);
2. s is an individual Web service in a service class, which resides on a specific server on the network;
3. l is one of the service levels provided by an individual Web service;
4. R is the constraint on the end-to-end delay.

For an individual service, we make the following definitions:

1. For a service s, the number of service levels it provides is $L(s)$;
2. Each service level guarantees a service time $e(s,l)$ time;
3. Each service level has a maximum capacity $C_{max}(s,l)$ on the number of clients it can accept;
4. Each service level has a current capacity $C_{cur}(s,l)$, the current number of clients in this level;
5. Each service level has a cost $c(s,l)$.

To make selection decisions, we define a *benefit* function based on the server load. The idea is that new service requests should select a server that has a light load. This is because a client selecting a server currently with a lighter load is more likely to experience a shorter response time. We thus use a benefit function to encourage such selections. Moreover, such selections will distribute user requests more evenly among all servers and create a globally balanced server loads. The benefit function $b(s,l)$ for selecting a service s at level l is a function of server load $\frac{C_{max}(s,l)-C_{cur}(s,l)}{C_{max}(s,l)}$, with the following properties:

1. $b(s,l)$ increases with increasing $\frac{C_{max}(s,l)-C_{cur}(s,l)}{C_{max}(s,l)}$;
2. $b(s,l)$ is continuous;
3. When the current number of clients in a service level is zero, it produces the maximum benefit, i.e. $b_{max}(s,l)=1$, when $C_{cur}(s,l)=0$;
4. When the current number of clients in a service level reaches the maximum capacity of this level, no benefit is produced, i.e. $b_{min}(s,l)=0$, when $C_{cur}(s,l)=C_{max}(s,l)$.

A user may define any benefit function as long as it satisfies the above criteria. For example, the benefit function may be a linear function or an exponential function of server load.

Since different services are provided by different service providers, they communicate by passing requests and data on the network. Transferring results from one service to another or to the user incurs some overhead. In this paper, we assume the transmission time and cost between any two services or between a service and a client are fixed and predefined. We include this network communication delay in our model.

Finally, one last constraint of the service selection is that the service selection for a new request should not disturb the activities of the current clients. So the service and service level selected by a broker must be among those currently available candidates i.e. $C_{max}(s,l)-C_{cur}(s,l)>0$.

4.2 Utility function

For sequential service composition with one QoS constraint, we model the end-to-end response time as the user's QoS constraint. Since users want to maximize the benefit they receive and minimize the cost they have to pay, we therefore define a utility function as a weighted sum of these two factors:

$$F(s,l) = w_b * \left(\frac{b(s,l) - avg_b}{std_b}\right) + w_c * \left(1 - \frac{c(s,l) - avg_c}{std_c}\right) \quad (1)$$

where

w_b,w_c	weights of the benefit and the cost ($0 < w_b,w_c < 1$, $w_b + w_c = 1$)
$b(s,l)$, $c(s,l)$	benefit and cost by choosing service s, level l
avg_b, avg_c	average benefit and cost for available services and service levels
std_b, std_c	standard deviation of benefit and cost for available services and service levels

We could include other attributes in the utility function and adjust the weight according to their importance. For example, we could add a reliability term and an availability term. The selection algorithms presented in this paper are not affected by the utility defined.

4.3 Pipeline structure and DAG structure

There many be several ways to construct a composite service or business process, each contains several individual Web services to be executed in sequential order. We call such way as an execution path of the composite service. If the composite service has only one execution path, it's structure is considered as a pipeline (see Fig. 2). If more than one execution paths exist, we consider that the composite service is structured as a DAG . For composite service shown in Fig. 3(*a*), it contains 5 paths shown in Fig. 3(*b*). Each path has a best overall utility and the path that produces the highest utility is the optimal solution for clients.

Fig. 2 Pipeline structure

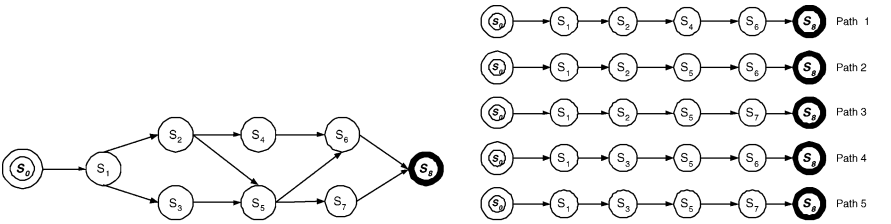

Fig. 3 DAG structure

5 Service selection algorithms

In this section, we investigate the service selection algorithms used by the QoS broker for sequential composite flow models with only one QoS constraint (i.e. response time). There are two approaches we can use to select the optimal services for each component of a business process. One is the *combinatorial approach*, by modeling the problem as a Multiple Choice Knapsack problem. The other is the *graph approach*, by modeling the problem as the constrained shortest path problem in the graph theory.

5.1 Combinatorial approach

The algorithms we present first can be used by composite services that are structured as a simple pipeline. However, it can also be used to find the execution path in a composite service that are structured as a DAG. In order to find the optimal path in a DAG-structured composite service, we need to run the algorithms several times, one for each possible execution path. We then choose the one with the highest utility as the best solution.

In this problem formulation, we assume the network connection overheads between any two services in the consecutive steps are the same. That is, if the data flow direction is service class $S_a \rightarrow S_b$(from a service in S_a to a service in S_b), we assume the network overhead between any service s_i in S_a and any service s_j in S_b are all the same. The transmission delay and cost between any two services are then added to the sending service. That is, the transmission overhead from s_i to s_j will be added to s_i. $c(s_i,l) = c(s_i,l) + c_{ij}$ and $r(s_i,l) = e(s_i,l) + d_{ij}$. The transmission delay and cost between the last service and the client will be added to the last service.

For a sequential composite service that has k steps (k service classes in an execution path) $(S_1,S_2,...,S_k)$, suppose the total response time constraint $\leq R$, the problem can be modeled as a Multiple-Choice Knapsack Problems (MCKP) which is defined as follows. Given a set of items in several classes

and a knapsack, where each item has a weight and profit, and the knapsack has a capacity, MCKP is to select one item from each class to be placed in the knapsack within the capacity yet has the highest total profit. Figure 4 illustrates the MCKP.

We can model the composite service selection problem as a MCKP in the following way:

1. The steps of the business process represents the classes in MCKP;
2. Since every service in a service class has many service levels, we consider each service level is a *candidate* for service selection; thus every candidate represents an item in that class;
3. The response time of each candidate represents the weight of the item in MCKP;
4. The utility a candidate produces represents the profit of the item in MCKP;
5. The objective is to maximize the total utility produced by the composite service under the constraint that the total response time $\leq R$;

The problem is thus formulated as:

$$\begin{aligned} &Max && \sum_{i=1}^{k} \sum_{j \in s_i} F_{ij} x_{ij} \\ &Subject\ to && \sum_{i=1}^{k} \sum_{j \in s_i} r_{ij} x_{ij} \leqslant R \\ & && \sum x_{ij} = 1, \quad i = 1, ..., k, \ j \in s_i \\ & && {}_{ij} \in \{0, 1\}, \qquad i = 1, ..., k, \ j \in s_i \end{aligned} \tag{2}$$

F_{ij} : *Utility value at step i for candidate j*

r_{ij} : *Response time of candidate j at step il*

T : *Total response time*

The MCKP problem is NP-hard. For large systems, it will be very difficult to always find the optimal solution. However, some preprocessing on the candidates of each class may reduce the number of candidates in each class. The following condition is presented in (Pisinger 1995):

If two items a and b in the same class S_i, satisfy

$$r_{ia} \leq r_{ib} \quad and \quad F_{ia} \geq F_{ib} \tag{3}$$

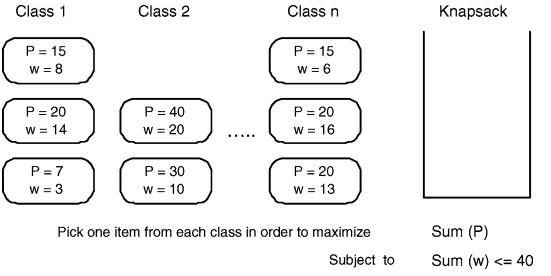

Fig. 4 MCKP

then an optimal solution to MCKP with $x_{ib}=0$ exists. We thus can delete item b from the candidate list in S_i.

In the following we will present three algorithms: exhaustive search, dynamic programming, and a minimal algorithm for MCKP.

5.1.1 Exhaustive search algorithm

This algorithm is to construct all service combinations and compares their utilities. It can always produce the optimal solution but is time and memory consuming. So it is only suitable when the number of classes and the items of each class are all small. For a composite service contains k steps and step i has $L(i)$ ($i=1,2,...,k$) candidates, the time complexity of the exhaustive search algorithm is $O(\prod_{i=1}^{k} L(i))$.

5.1.2 Dynamic programming algorithm

The MCKP problem can be solved in pseudo-polynomial time using dynamic programming. Given a pair of integers l ($1 \le l \le k$) and $\hat{c}$ ($0 \le \hat{c} \le T$), consider the sub-instance of MCKP consisting of subsets $S_1,S_2,...,S_l$ and capacity $\hat{c}$. Let $f_l(\hat{c})$ denotes its optimal solution value. The problem can be solved by the following dynamic programming formulation:

$$\text{Let } \bar{r}=\min\{r_j, j\in S_i\}\ i=1,2,...,k$$

$$f_1(\hat{c})=\begin{cases}-\infty & \hat{c}=0,1,...,\bar{r_1}-1\\ \max\{F_{1j}: j\in S_1, r_j\le\hat{c}\} & \hat{c}=\bar{r_1},...,R\end{cases}$$

$$f_l(\hat{c})=\begin{cases}-\infty & \hat{c}=0,1,...,\sum_{k=1}^{l}\bar{r_k}-1\\ \max\{f_{l-1}(\hat{c}-r_j)+F_{lj}: j\in S_l, r_j\le\hat{c}\} & \hat{c}=\sum_{k=1}^{l}\bar{r_k},...,R\end{cases} \quad 2\le l\le k \tag{4}$$

The optimal solution is the state corresponding to $f_k(T)$. For a composite service contains k steps and step i has $L(i)$ ($i=1,2,...,k$) candidates, the time complexity of the dynamic programming algorithm is $O(\sum_{i=1}^{k} L(i)*R)$.

5.1.3 Pisinger's algorithm

In (Pisinger 1995), Pisinger introduces an algorithm for efficiently solving the MCKP problem. This algorithm first solves the linear MCKP (LMCKP) problem by using a partitioning algorithm and derives an initial feasible solution (*initial core*) to MCKP. It then uses dynamic programming to expand the initial core by adding new classes as needed. The algorithm is outlined below.

1. Solving Linear Multiple-Choice Knapsack Problem (LMCKP) In Eq. (2), if the integrity constraint $x_{ij}\in\{0,1\}$ is relaxed to $0 \le x_{ij} \le 1$, the problem

becomes LMCKP. In Zemel (1984) and Dyer (1984), Zemel and Dyer each developed linear time algorithms for LMCKP. Both algorithms are based on the convexity of the LP-dual problem to Eq. (2). For the dual problem, we can pair the dual line segments and delete the unpromising ones according to some dominance criteria.

Based on Dyer and Zemel's algorithms, Pisinger (1995) presents a partitioning algorithm to solve the LMCKP. The optimal solution x^* to LMCKP is composed by the LP-optimal choices b_i in each class, where $x_{ibi}=1$. One of the classes S_a may contain two non-zero fractional variables x_{ab_a} and $x_{ab'_a}$ $(x_{ab_a}+x_{ab'_a}=1)$. If x^* has no fractional variables, it is already the optimal solution to MCKP. Otherwise, the fractional class S_a is defined as the *initial core* for MCKP. It then continues the second step.

2. Solving MCKP from LMCKP Given an initial core and the set of $\{b_i | i \neq a\}$ from step 1, the positive and negative gradient λ_i^+ and λ_i^- for each class $S_i, i \neq a$ are defined as:

$$\lambda_i^+ = \max_{j \in S_i, r_{ij} > r_{ib_i}} \frac{F_{ij} - F_{ib_i}}{r_{ij} - r_{ib_i}}, i = 1, 2, \ldots, k, i \neq a, \tag{5}$$

$$\lambda_i^- = \max_{j \in S_i, r_{ij} < r_{ib_i}} \frac{F_{ib_i} - F_{ij}}{r_{ib_i} - r_{ij}}, i = 1, 2, \ldots, k, i \neq a, \tag{6}$$

The algorithm then sorts the sets $L^+ = \{\lambda_i^+\}$ in decreasing values, and $L^- = \{\lambda_i^-\}$ in increasing values. Starting from the initial core, it will expand the core by alternately including a new (not yet selected) class S_i that has the largest λ_i^+ from L^+ or the smallest λ_i^- from L^-. When all classes have been added to the core, the optimal solution for MCKP is found.

The computational experiments in Pisinger (1995) show that this algorithm is usually very efficient and much faster than the dynamic programming algorithm. For a composite service contains k steps (k classes, $S_1,..,S_k$) and step i has $L(i)$ $(i=1,2,...,k)$ candidates, the time complexity of Pisinger's algorithm is $O(\sum_{i=1}^{k} L(i) + R * \sum_{S_i \in C} L(i))$ is the core.

In the worst case, the time complexity of Pisinger's algorithm is the same as dynamic programming. However, as we will show later in this paper, Pisinger's algorithm usually converges very fast, allowing its computation time to be much less than dynamic programming.

Example Now we present an example of Pisinger's algorithm used to solve a sequential composite service. The example is defined as follows.

– The benefit function $b(s,l)$ is defined as:

$$b(s,l) = \frac{1 - e^{-\frac{C_{max}(s,l) - C_{cur}(s,l)}{C_{max}(s,l)}}}{1 - e^{-1}} \quad 0 \leq C_{cur}(s,l) \leq C_{max}(s,l) \tag{7}$$

– The end-to-end response time constraint $R \leq 61$;
– The composite service contains 4 sequential services ($k=4$), each has 4 service levels ($n=4$). We assume all service levels are available, i.e.

Table 4 Response time and cost (response time, cost) = ($r(s,l)$, $c(s,l)$)

	Level 1	Level 2	Level 3	Level 4
Service 1	(3, 33)	(12, 28)	(21, 19)	(30, 13)
Service 2	(4, 24)	(10, 19)	(18, 14)	(26, 8)
Service 3	(7, 25)	(18, 20)	(30, 15)	(36, 10)
Service 4	(9, 26)	(17, 21)	(24, 16)	(33, 11)

Table 5 Max and current capacity (Max, Current) = ($C_{max}(s,l)$, $C_{cur}(s,l)$)

	Level 1	Level 2	Level 3	Level 4
Service 1	(20, 16)	(34, 4)	(34, 12)	(44, 31)
Service 2	(15, 4)	(13, 2)	(12, 1)	(7, 1)
Service 3	(37, 13)	(31, 13)	(29, 7)	(15, 1)
Service 4	(24, 7)	(20, 2)	(53, 11)	(10, 4)

$C_{max}(s,l) - C_{cur}(s,l) > 0$. The response time, cost, maximum capacity and current capacity of each service and service level are shown in Tables 4 and 5.

According to the definition of the benefit function Eq. (7) and utility function Eq. (1), we can get the utility of each service level. We then convert the utility to non-negative integers using the formula $F(s,l) = floor(F(s,l)*200) + min(F(s,l))$. The converted utility is shown in Table 6. Also, we use the pre-processing criteria 3 to delete some items in each class and get the final $r(s,l)$ & $F(s,l)$ shown in Table 7. Now we can model the problem as a MCKP shown in Figure 5.

When using Pisinger's algorithm, we first solve the corresponding LMCKP to get the initial solution to MCKP. The initial core contains items: {7[3],18[18],30[136],36[241]}

$$
\begin{aligned}
&S_1: \quad b_1 = 2, \quad r_{1b_1} = 12, \quad F_{1b_1} = 167;\\
&S_2: \quad b_2 = 3, \quad r_{2b_2} = 18, \quad F_{2b_2} = 232;\\
&S_3: \quad \textit{fractional class}, \quad \textit{initial core}\\
&S_4: \quad b_4 = 2, \quad r_{4b_4} = 17, \quad F_{4b_4} = 143.
\end{aligned}
$$

Starting from the initial core, we calculate the positive and negative gradient λ^+ and λ^- for each class S_i, $i \neq 3$ and sort $L^+ = \{\lambda_i^+\}$ according to decreasing values (Table 8), and $L^- = \{\lambda_i^-\}$ according to increasing values (Table 9).

Table 6 Utility Utility $F(s,l)$

	Level 1	Level 2	Level 3	Level 4
Service 1	10	167	191	162
Service 2	10	140	232	240
Service 3	3	18	136	241
Service 4	16	143	143	96

Table 7 Response time and corresponding utility Response time $r(s,l)$[Utility $F(s,l)$]

	Level 1	Level 2	Level 3	Level 4
Service 1	3[10]	12[167]	21[191]	–
Service 2	4[10]	10[140]	18[232]	26[240]
Service 3	7[3]	18[18]	30[136]	36[241]
Service 4	916]	17[143]	–	–

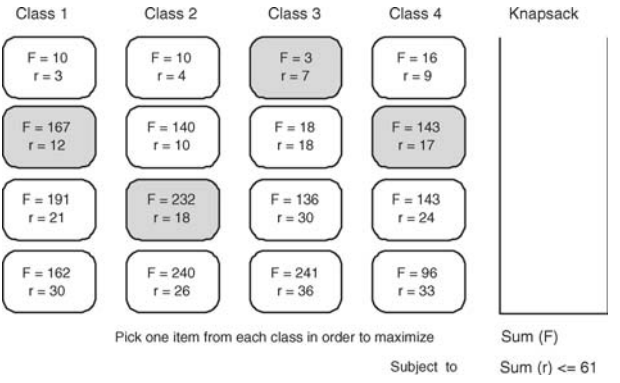

Fig. 5 MCKP example

Table 8 L^{+} set

No.	dr[dF]	Class {Items(r[F])}
1	9[24]	S_1\{12[167],21[191],3[10]\}
2	8[8]	S_2\{18[232],26[240],4[10],10[140]\}
3	1[0]	S_4\{17[143],9[16]\}

Table 9 L^{-} set

No.	dr[dF]	Class{Items(r[F])}
1	9[24]	S_1\{12[167],21[191],3[10]\}
2	8[8]	S_2\{18[232],26[240],4[10],10[140]\}
3	1[0]	S_4\{17[143],9[16]\}

The set of partial vectors $Y_C = \{(\mu, \pi_i, \upsilon_i)\}$ in the initial core C has 4 states: $Y_C = \{(54,545,0), (65,560,1), (77,678,2), (83,783,3)\}$. After performing the state reduction according to the upper bound test see Pisinger (1995), Y_C becomes $Y_C = \{(54, 545,0), (65, 560,1)\}$. In this Y_C , (54, 545,0) is chosen with item (7[3]) and utility $z = 545$. In the following steps, we will add new classes to the core until all classes have been considered.

1. Add class S_1\{12[167],21[191],3[10]\} from L^+ to the core, $Y_C = \{(54,545,0), (63,569,1)\}$; The item chosen in S_1 is 12[167], the utility $z = 545$;
2. Add class S_2\{18[232], 26[240],4[10],10[140]\} from L^- to the core, $Y_C = \{(54,545,1), (63,569,3)\}$; the item chosen in S_2 is 18[232], the utility $z = 545$;
3. Add class S_4\{17[143],9[16]\} from L^- to the core, $Y_C = \{(63,569,3)\}$; The item chosen in S_4 is 17[143], the utility $z = 545$;

At this point, the core is complete. The optimal solution includes the selected item in each class: {12,18,7,17} and the maximum utility value is $z = 545$.

5.1.4 Performance study

We have conducted many tests using the three algorithms to compare their running time. Table 10 shows the test result. Since the exhaustive search algorithm is time and memory consuming, it is only suitable when both k (service stages of the composite service) and n (number of candidates in each stage) are small (In our tests, we choose $k + n \leq 25$). When k and n become larger, this algorithm quickly ran out of memory in our experiments. Between dynamic programming and Pisinger's algorithm, Pisinger's algorithm is much faster and more efficient than dynamic programming. For example, when $k = 10$ and $n = 100$, it takes dynamic programming 35,668μs to finish the computation while Pisinger's algorithm only needs 217μs.

The test result shows that Pisinger's algorithm is the best algorithm for service selection in composite service, especially when the number of candidates in each service is large.

Table 10 Run time comparison

Service (k)	Candidate(n)	Running time (μs)		
		Exhaustive search	Dynamic programming	Pisinger's
5	5	127	99	56
5	50	–	4649	116
6	5	1131	130	37
10	10	–	1395	88
10	100	–	35668	217
10	1000	–	778990	1701
20	100	–	289071	304
20	1000	–	7 *s* 67617 μs	3240
50	100	–	1 *s* 830409 μs	685
50	1000	–	26 *s* 928142 μs	4778
100	100	–	6 *s* 273286 μs	1294

5.2 Graph approach

Graph algorithms can be used to handle various composite services, whether they are structured as pipelines or DAG's. The algorithms only need to be run once to find the optimal solution. Our proposed algorithms are based on the shortest path algorithms that are commonly used in the graph theory. We modify the classical shortest path algorithms to handle delay constraints. We present a constrained Bellman-Ford algorithm (Widyono 1994) and a constrained shortest path algorithm (CSP).

5.2.1 Graph construction

Each service level in every individual service is represented by a node in the graph, with a cost and a benefit value. However, the classical shortest path algorithms designed for graphs usually define costs on edges, not on nodes. To use algorithms like Bellman-Ford, we need to transform a composite service graph into a service path graph so that edges, not nodes, have weights (costs, delays and etc.). The graph can be constructed as follows.

1. Each service level of each individual service is represented as a node in the graph;
2. If service s_i is connected to service s_j, all service levels in s_i are connected to all service levels in s_j;
3. Set link cost, delay and benefit: To remove parameters from graph nodes, we add the service time and cost of the receiving node to all incoming links to the node. That is, if data is sent from s_i to s_j, we will add service time $e(s_j,l)$ and cost $c(s_j,l)$ to the link connected to s_j. So for the link from service level l_a of s_i to service level l_b of s_j , its delay is set to $d_{ij}+e(s_j,l_b)$ and its cost is $c_{ij}+c(s_j,l_b)$. The benefit of the link is set to $b(s_j,l_b)$;
4. Suppose there are k service classes in the execution plan, we add a source node (S_0) and a sink node (S_{k+1}) to the graph as shown in Figure 6. For all nodes that have no incoming edges, add links from the source node to them; the delay, cost and benefit of these links are set to 0. For all vertexes that have no outgoing edges, connect them with the sink node. The delay and cost of these links are set to be the transmission delay and cost

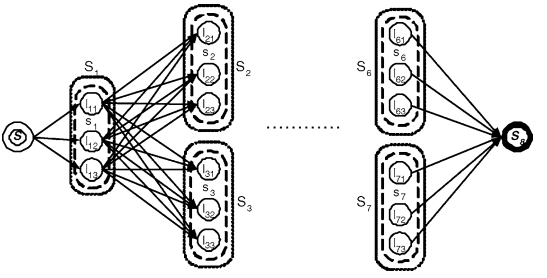

Fig. 6 Constructed DAG

between the service (that provides the start node of the link) and user. The benefit of these links are set to 0;

5. For each link, according to Eq. (1), we can compute its utility F based on the benefit and cost of the link.

After these steps, we have a DAG, in which every link has a delay and a utility. The service selection problem now can be defined as:

Find a path that produces the highest utility between source S_0 ***and sink*** S_{k+1} ***subject to the delay constraint*** R.

5.2.2 Constrained Bellman-Ford (CBF) algorithm

In (Widyono 1994), the CBF algorithm has been used to find the path with the minimum cost between the source and the destination subject to the delay constraint. We can use this algorithm to solve our problem. Instead of searching for the minimum cost path, we search for the highest utility path. The algorithm uses a breadth-first search, discovering paths of monotonically increasing delay and updating the highest utility path to every node it has visited. Detailed description of CBF can be found in (Widyono 1994). The running time of CBF grows exponentially.

5.2.3 Constrained Shortest Path (CSP) algorithm

In CBF, all paths are searched using the breadth-first approach. However, since the composite service graph is a DAG, we can modify the classical DAG shortest path algorithm to a more efficient constrained shortest path algorithm to solve the problem. The idea is that we can first topologically sort all nodes in the graph, then visit each node in the topological order and "relax" it. In each node, we maintain a list containing the utility gained for different paths from the source node to it under the delay constraint. The list

Table 11 Algorithm 1

```
//every node contains a list (list(node))of paths
Sort every node's outgoing edges in increasing delay;
Topologically sort the nodes in graph G;
For each node i, in topological order
  For each outgoing edge of i, in increasing delay order
    j = end_node;
    delay = edge.delay; utility = edge.utility;
   If (i = = source) Relax(i,j,delay,utility);
   Else for (each path p in list(i))
     delay = delay + p.delay; utility = p.utility;
    If (delay ≤ constraint) Relax(i,j,delay,utility);
// Relax: Add path to end node path list
Relax(source,end,delay,utility)
    if new-path np is dominated by an old-path return;
    if any old-path p is dominated by np
    remove p from link(end)); Add np to link(end)
```

is sorted in monotonically decreasing order of utility. The relax function used in our algorithm is defined by the *dominate* relationship: given two paths a and b, if ($a.utility \geq b.utility$) and ($a.delay \leq b.delay$), then we say b is dominated by a. The relax function adds a path to the list only when no other path in the list dominates it. When a new path is added and if there is any existing path in the list dominated by the new path, the old path will be removed. After we visit all nodes, the highest utility in the sink node is the solution for the selection problem. The constrained shortest path (CSP) algorithm is defined in Table.

5.2.4 Performance study

As discussed, there are three algorithms to find an optimal solution in composite service structured as a DAG.

1. All Execution Paths (AEP) algorithm. In this algorithm, we first use a breadth-first-search to find all execution paths in the execution plan. Second, for each execution path, it use combinatorial approach based algorithm, such as Pisinger's algorithm, to find the best utility of this path. Finally, it compares the utilities of all paths and pick one that produce the highest utility as the optimal solution.
2. CBF algorithm
3. CSP algorithm

For CBF and CSP, we treat the whole system as a graph and search for the highest utility path in the graph under the end-to-end delay constraint.

Table 12 compares the running time of the three algorithms. We generated k service classes , each service class has n candidates, and a service class is connected to at most b service classes (branch).

Table 12 Running time comparison for DAG

Test cases				Running time (μs)		
branch(b)	service class(k)	candidate(n)	execution paths(ep)	AEP	CBF	CSP
2	5	5	3	564	1,377	1,040
2	10	5	8	1,520	82,562	9,382
2	10	10	12	2,628	554,732	131,353
2	10	50	24	17,136	–	1 *s* 758,156
2	20	10	72	21,816	–	457,371
2	20	50	144	129,600	–	7,507,685
3	10	5	10	1,900	4,130	1,794
3	10	10	10	2,190	171,196	73,103
3	10	50	20	14,280	–	690,372
3	10	100	10	13,580	–	424,052
3	20	10	19	5,757	18,467,446	252,886
3	20	50	49	44,100	–	3,796,061
4	20	10	31	9,393	12,758,279	288,958
4	50	10	108	156,600	–	390,931
5	100	10	297	547,074	–	1,579,118

The testing result shows CSP is much better than CBF, whose running time grows exponentially. All empty entries in the table represent very long executions that were aborted before completion. As the graph becomes large, CBF takes about 60-100 times longer than CSP. But even CSP algorithm is slow compared to AEP. As the numbers of services and candidates increase, CSP grows faster than AEP. In the rare cases when the number of candidates for each service is small, CSP may have a better performance than AEP.

Although the test result shows AEP runs faster than CSP, we need to be aware of its limits. The problem of AEP is that it does not model network transmission delays and costs accurately. Since AEP is based on the multiple choice knapsack problem, it uses the same value for transmission delays and costs between all services in two adjacent service classes. As mentioned before, different services from different service providers may reside in different locations. Services in two adjacent service classes may be connected by different networks, some with long and unpredictable delays or high delivery costs. Since AEP models the problem as a knapsack problem, its strength is on the combinatorial selection of services, not the flow of a graph. Therefore, Pisinger's algorithm cannot differentiate different service locations and treat them differently. This will affect the selection result if the network cost between two consecutive services is not uniform.

On the other hand, CSP and CBF solve this problem nicely since they are designed for the graph model. Every link that connects services (or service levels) in two adjacent service classes can be treated differently and assigned different values on network transmission delay and cost. They will be more precise than AEP and the selection result should be more reliable even though they take more time in large services cases.

The above discussion can be seen from Fig. 7. Suppose service class S_1 is connected to service class S_2. S_1 has two service candidates s_1 and s_2, S_2 has service candidates s_3 and s_4. Each service has two service levels $l_{i1}$and l_{i2} ($i=1,2,3,4$), and each service level has a service time e_{ij}, cost c_{ij} and benefit $b_{ij}(i=1,2,3,4;\ j=1,2)$. The network transmission delay ($d_{ij}(i=1,2;\ j=3,4)$) and cost ($c_{ij}(i=1,2;\ j=3,4)$) between these services are shown in Fig. 7.

For CSP, we can add the service time and cost of the receiving node (a service level) to incoming links. So the links between different services can be

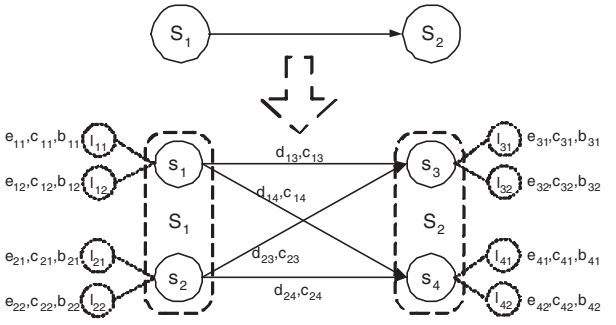

Fig. 7 Composite service connection

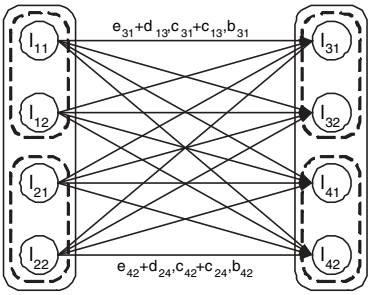

Fig. 8 CSP cost model

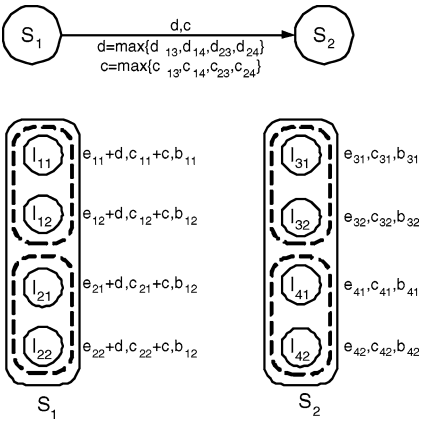

Fig. 9 MCKP cost model

treated separately. As shown in Fig. 8, the link from service level 1 in Service 1 to service level 1 in Service 3 has a delay of $d_{13}+e_{31}$, cost of $c_{13}+c_{31}$, and benefit b_{31}. The delay, cost and benefit of the link from service level 2 in Service 1 to service level 2 in Service 4 are $d_{14}+e_{42}$, $c_{14}+c_{42}$ and b_{42}, respectively. For each link, we can use Eq. (1) to calculate its utility. Now, the delay, cost and benefit of the nodes have been transfered to links and we can use CSP to solve the shortest path problem.

For AEP, in Fig. 9, the links between all services in two adjacent service classes are treated equally even when they may have different overheads. So for the system shown in Fig. 7, we derive the transmission delay and cost from S_1 to S_2 as: $d=max\{\ d_{13},\ d_{14},\ d_{23},\ d_{24}\}$ and $c=max\{\ c_{13},\ c_{14},\ c_{23},\ c_{24}\}$. And all network overheads are added to the sending node . So the service levels in service s_1 and s_2 now have the response time and cost as:$r_{ij}=e_{ij}+d$, $c_{ij}=c_{ij}+c$ ($i=1,2$; $j=1,2$). In this way, the network transmission delay and cost are transfered to the nodes.

The decision of whether to use AEP or CSP depends on whether the network overheads will be very different in the system. If service candidates are distributed over a wide area (such as from different countries or across

the nation), CSP will be able to take their connection cost into account and present better solutions. For systems where all candidates are in the same network area, such as those in an intra-enterprise network, AEP is faster and can present very good solutions.

6 Conclusions

In this paper, we study the end-to-end QoS constraint issue for composite business processes that are built using the Web service framework. The QoS guarantee is provided by a QoS broker that is responsible for coordinating among composed services to meet the quality constraint for a client. We have presented service selection algorithms. The objective of the algorithms is to maximize user-defined service utilities while meeting the end-to-end performance constraint. We propose two approaches to solve the problem. One is the combinatorial approach, by modeling the problem as a MCKP. The other is the graph approach, by modeling the problem as a constrained shortest path problem in the graph theory. We have presented efficient algorithms for both approaches. The performance of these algorithms have been studied and compared. The selection of algorithms depends on the problem size, network structure and other factors. Both algorithms have a reasonable run time even for problems with a large data set. We believe the proposed models and algorithms present practical solutions to the end-to-end QoS problem for composite Web services.

References

Ankolekar A, Burstein M, Hobbs JR, Lassila O, Martin DL, McIlraith SA, Narayanan S, Paolucci M, Payne T, Sycara K, Zeng H (2001) DAML-S: Semantic markup for Web services. Proc. of the International Semantic Web Working Symposium, Stanford, CA

Arkin A, Askary S, Fordin S, Jekeli W, Kawaguchi K, Orchard D, Pogliani S, Riemer K, Struble S, Takacsi-Nagy P, Trickovic I, Zimek S (2002) Web service Choreography Interface (WSCI) 1.0, http://www.w3.org/TR/wsci/

BPMI org (2002) Business Process Modeling Language (BPML). Version 1.0, http://www.bpmi.org/bpml.esp

Casati F, Ilnicki S, Jin L, Krishnamoorthy V, Shan M (2000) Adaptive and dynamic service composition in eflow. Technical Report, HPL-2000

Chen H, Yu T, Lin KJ (2003) QCWS: An Implementation of QoS Capable Multimedia Web Services. Proc. of IEEE 5th Int. Symp. Multimedia Software Engineering, Taiwan

Christensen E, Curbera F, Meredith G, Weerawarana S (2001) Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl

Curbera F, Goland Y, Klein J, Leymann F, Roller D, Thatte S, Weerawarana S (2003) Business Process Execution Language for Web services, Version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel

Dyer ME (1984) An O(n) algorithm for the multiple-choice knapsack linear program. Mathematical Programming 29:57–63

Gudgin M, Hadley M, Mendelsohn N, Moreau JJ, Nielsen HF (2003) Simple object access protocol (SOAP) 1.2, http://www.w3.org/TR/soap12

Leymann F (2001) Web service flow language (WSFL) 1.0. http://www-.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Martello S, Toth P (1990) Knapsack Problems, Algorithms and Computer Implementations. John Wiley & Sons Ltd

OASIS Universal Description. Discovery and Integration Specification TC, http://www.oasis-open.org/committees/uddi-spec/

Patel C, Supekar K, Lee Y (2003) A QoS Oriented Framework for Adaptive Management of Web service based Workflows Database and Expert Systems (DEXA-2003) conference. Prague, Czech Republic

Pisinger D (1995) A minimal algorithm for the Multiple-choice Knapsack Problem. European Journal of Operational Research 83:394–410

Ponnekanti SR, Fox A (2002) Sword: A developer toolkit for Web service composition, 11th World Wide Web Conference (Engineering Track), Honolulu, Hawaii

Shankar M, DeMiguel M, Liu JWS (1999) An end-to-end QoS management architecture. Proc. 5th Real-Time Technology and Applications Symposium

Sheng QZ, Benatallah B, Dumas M, Mak E (2002) SELF-SERV: A Platform for Rapid Composition of Web services in a Peer-to-Peer Environment. Proc. of the 28th Very Large DataBase Conference (VLDB'2002), Hong Kong, China

Thatte S (2001) XLANG: Web services for business process design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

The Oceano project, IBM. http://www.research.ibm.com/oceanoproject/index.html

VanderMeer D, Datta A, Dutta K, Thomas H, Ramamritham K, Navathe SB (2003) FUSION: A System Allowing Dynamic Web service Composition and Automatic Execution. Proc. of IEEE International Conference on E-Commerce, Newport Beach, CA, USA

Widyono R (1994) The design and evaluation of routing algorithms for real-time channels. Tech. Rep. TR-94-024, University of California at Berkeley, International Computer Science Institute

Zemel E (1984) An O(n) algorithm for the linear multiple choice knapsack problem and related problems. Information Processing Letters 18:123–128

Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ (2003) Quality Driven Web Service Composition. Proc. 12th International World Wide Web Conference (WWW)