An Adaptive Service Selection Approach to Service Composition *

Lijun Mei
The University of Hong Kong
Pokfulam, Hong Kong
ljmei@cs.hku.hk

W.K. Chan †
City University of Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

T.H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

Abstract

In service computing, the behavior of a service may evolve. When an organization develops a service-oriented application in which certain services are provided by external partners, the organization should address the problem of uninformed behavior evolution of external services. This paper proposes an adaptive framework that bars problematic external services to be used in the service-oriented application of an organization. We use dynamic WSDL information in public service registries to approximate a snapshot of a network of services, and apply link analysis on the snapshot to identify services that are popularly used by different service consumers at the moment. As such, service composition can be strategically formed using the highly referenced services. We evaluate our proposal through a simulation study. The results show that, in terms of the number of failures experienced by service consumers, our proposal significantly outperforms the random approach in selecting reliable services to form service compositions.

Keywords*: service selection, service composition, quality, link analysis, adaptive framework*

1. Introduction

A service is business functionality with well-defined interface. Software applications using *service-orientation* are increasingly popular. In this paradigm, individual services publish their functionality interface in registries (such as UDDI registries). Based on the content of a registry, a service consumer discovers a potential service provider and then loosely enacts with the selected service. A resultant application is called a *service composition*.

On one hand, the quality of individual services [6][7][11] should play an important role in offering useful service composition [1][12][13] for service-oriented (SO) applications. On the other hand, not all services can be under the control of a single organization. In the cross-organizational environment, an SO application may use a service provided by another organization. For instance, when an e-shop needs to charge customers certain fees via their credit cards, the de facto approach is to use the services provided by, say, Visa or MasterCard directly rather than having to develop an in-house service. (Indeed, some customers of the e-shop may be concerned about privacy and are skeptical to use any in-house credit card billing service.) Fault handling in the local organization's business process cannot turn a failed credit card service into a successful one. A failure of such an external service will thus be directly observable by a customer of the SO application.

Unlike in-house services, the number of external services suitable for a particular business purpose can be large. For example, there are numerous financial news providers worldwide (such as cnn.com and mpinews.com). Hence, if an SO application wants to use an external service as one of its business workflow steps, an important challenge is to be able to select one with the highest quality. Objective service ranking would be attractive.

The service ranking problem is not totally new. For instance, Gekas and Fasli [14] proposed to apply a network analysis mechanism for the semantic web. They firstly captured the link between any two services based on the semantic conformance of input/output data types for these two services, and then used all the links to form a graph to apply link analysis. Nevertheless, defining the link according to interface parameters may not necessarily be the usual criterion for ranking services that represent business functionality. Let us take the credit card service again as an illustration: The standard parameters for VISA card services from different banks are likely to be the same. However, the qualities of credit services provided by banks vary more at the business reputation level than at the technical interface level.

We however observe that a service provided by a large and reputable bank (or the service of PayPal) is more likely to be used by a potential customer. For an SO application to invoke these reputable services, it can specify the service address in the WSDL documents. Based on the above observation, this paper proposes an adaptive framework to identify reliable external services for service-oriented applications in the cross-organizational environment. In this framework, a consumer P using a service Q can count locally the number of times that Q

* This research is supported in part by the General Research Fund of the Research Grant Council of Hong Kong (project nos. 111107, 716507, and 717506).

† Corresponding author.

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.22

successfully serves *P*, and the number of failure cases alike. We then use the local snapshot of the public service registries to model the network of services *N* that are visible to *P*. Based on the counts, the network *N* will be dynamically updated, such as by the removal of a link due to a recent experience of a failure when using *Q*.

Through the application of a link analysis algorithm on *N*, the changes in individual service selections are propagated to the entire snapshot of the network. We thus compute the new popularity measures of various services in the *global picture*, and only present the highly referenced services to the SO application for selection.

The main contributions of this paper are three-fold: (i) We propose a framework to select and compose services adaptively. (ii) We utilize the framework to present an algorithm to rank services using social network analysis. (iii) We present the first set of experiments to evaluate an adaptive approach to selecting services in an SOA environment. The experimental results show that our approach is promising.

The rest of the paper is organized as follows: Section 2 gives a motivating example to illustrate the challenges of service selection. Section 3 presents the preliminaries that lay the foundations of our framework. Section 4 introduces our framework to facilitate adaptive service composition, and details our framework by proposing an algorithm to rank services adaptively. Section 5 reports on the experiment that evaluates our approach. It is followed by a literature review and the conclusion in Sections 6 and 7, respectively.

2. Motivating Example

In this section, we present an example adapted from the *TripHandling* application [21] to motivate our work.

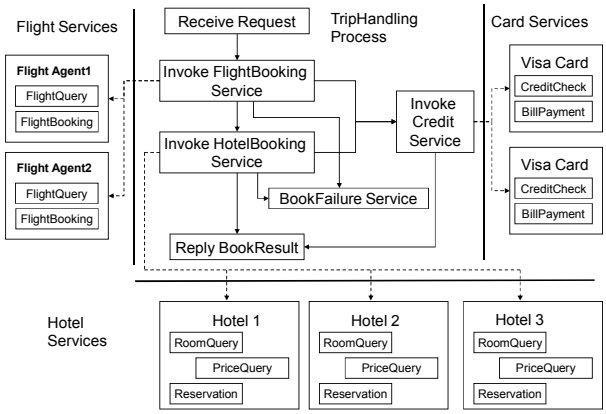

Figure 1. *TripHandling* application.

The *TripHandling* application handles requests from service customers by scheduling trips. It includes the following workflow steps: (i) *FlightBooking* to book flights for a trip, (ii) *HotelBooking* to book hotels for a trip, and (iii) *BillPayment* to bill the corresponding customers via an online credit card service for successful flight and/or hotel booking. If both *FlightBooking* and *HotelBooking* are successfully completed, the booking results will be sent to the customer. On the other hand, if either *FlightBooking* or *HotelBooking* encounters an error, the *TripHandling* application will terminate and display an error message. The structure of the *TripHandling* application is shown in Figure 1, in which the solid undirected lines separate the regions for different categories of services, the solid arrows links up the sequence of the workflow steps, and a dashed arrow means a potential service invocation.

In the following, we present a number of scenarios to illustrate the challenges in service composition.

Scenario 1: Inadequacy of Pre-Deployment Information. Suppose that *Hotel 1* is initially evaluated by *TripHandling* to be the preferred choice for providing hotel room availability query service, and *TripHandling* indeed binds to *Hotel 1* in its service composition as specified in the WSDL document. Usually, in such a situation, neither the potential (test) evaluations of *Hotel 2* and *Hotel 3* nor their potentials to bind with *TripHandling* will be presented in the WSDL documents. Suppose also that, after the service deployment, for some reason, the hotel room availability query service provided by *Hotel 1* becomes inaccessible. The *TripHandling* application will then be unable to provide useful *HotelBooking* services to customers. What if *Hotel 2* is available to provide such a query service? If *TripHandling* switches to use *Hotel 2* for *HotelBooking*, it may offer its services as usual. From this scenario, we know that selecting a service based purely on pre-deployment information is generally inadequate to avail a sustained service composition to customers.

Although a dynamic service discovery approach may obviously complement a static approach, our scenarios below show that there are other challenges in dynamic approaches.

Scenario 2: To Switch or Not To Switch, That is the Question. Suppose the chances of encountering a failure during service invocation (dubbed *failure rates*) for *Hotel 1* and *Hotel 2* are 2% and 5%, respectively. When the services of *Hotel 1* are inaccessible, *Hotel 2* can be used as the "backup" to replace *Hotel 1* as the provider of hotel room availability query service. However, because the failure rate of *Hotel 2* is much higher than that of *Hotel 1*, when the latter resumes its services, the system had better switch back to *Hotel 1* as the service provider. A simple strategy is to switch back to *Hotel 1* whenever it is available. On the other hand, *Hotel 1* may shut down again shortly afterwards. If *TripHandling* uses the simple strategy to compose its services, its customers may immediately experience a failure. An alternative is to wait for a sufficiently long period before switching back to *Hotel 1*. However, the longer the waiting period, the poorer will be the overall service quality, because the services from *Hotel 2* has a higher failure rate than those from *Hotel 1* (assuming all other factors to be equal). The proper strategy to switch back from *Hotel 2* to *Hotel 1* after the latter has resumed its services should be thoroughly considered.

Scenario 3: Evolving Quality. Services like Google search may upgrade themselves frequently. The failure

rates of *Hotel 1* and *Hotel 2* may evolve over time as well. In Scenario 2, we have hypothesized that *Hotel 1* has a lower failure rate than *Hotel 2*. Suppose that, as time goes on, the failure rates of *Hotel 1* and *Hotel 2* change to 2% and 1%, respectively. In Scenario 1, the pre-deployment information has shown that *Hotel 1* is the best provider of hotel room availability query service. Since *Hotel 2* has upgraded its service, it should be the preferred choice for hotel query service.

Suppose *TripHandling* uses *Hotel 1* without problem throughout its history. A challenging question is: How can a technique guarantee that it will eventually switch to *Hotel 2* for the dynamic quality improvement despite the successful experience with *Hotel 1*?

In summary, proper switching among individual services with evolving qualities is a challenging task in service composition. In this paper, we shall present our proposal to address these challenges.

3. Preliminaries

This section reviews the preliminaries that lay the foundations of our framework.

3.1. Link Analysis

The link analysis of a graph is a process to find the graph properties according to the link distributions among the nodes of the graph. The technique has been applied to the ranking of webpages in Internet search. There are various link analysis ranking algorithms [3], such as PageRank, InDegree, Hits, and Salsa. Since PageRank is the most representative technique in webpage ranking, we shall use it to illustrate how link analysis is conducted.

PageRank [19] computes the ranking of searchable pages based on a graph of the Web and is an integral part of Google search [20]. Intuitively, a document is ranked high by PageRank if it is linked to many other highly ranked documents. PageRank assumes that a Web user will eventually stop clicking any link. The probability, at any step, that the user will continue is known as a damping factor $d \in [0, 1]$, which is typically set to about 0.85 (see [5]). PageRank with the damping factor is defined as follows: Suppose a webpage u points to a set $F(u)$ of other pages, and u is also pointed to by a set of pages $B(u)$. Let us denote the size of $F(u)$ by $C(u)$. The page rank $PR(u)$ of the page u is given by

$$PR(u) = (1-d) + d \times \sum_{v \in B(U)} \frac{PR(v)}{C(v)} \quad (1)$$

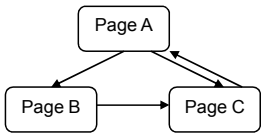

Figure 2. Example to illustrate web link analysis.

Figure 2 shows a graph of three webpages A, B, and C linked among one another. Based on equation (1), we can establish a set of recurring equations for these three pages with $d = 0.85$. By initially setting each of $PR(A)$, $PR(B)$, and $PR(C)$ to 1, we can iteratively re-compute the results until they converge.

$$PR(A) = 0.15 + 0.85 \times PR(C)$$
$$PR(B) = 0.15 + 0.85 \times (PR(A) / 2)$$
$$PR(C) = 0.15 + 0.85 \times (PR(A) / 2 + PR(B))$$

Solving the set of equations through 20 iterations, we obtain $PR(A) \approx 1.163$, $PR(B) \approx 0.6444$, and $PR(C) \approx 1.192$. It is known that the initial PageRank values will make the actual final scores of the pages different, but the ranks are generally believed to largely reflect the relative importance of the pages in the graph. PageRank is not optimal, and its improvement is still an active research area.

Since applying the link analysis algorithm can be expensive, the depth can be constrained to reduce the cost of computation. The solution, however, will be less precise.

3.2. Model of Service-Orientation

There are three distinct activities in a typical model of service-orientation, namely service registration, discovery, and binding. A service provider *registers* itself in a service registry. When the consumer wants to use the service, it must firstly *discover* the service from the registry, and then *bind* to the service. The service provider, service registry, and service consumer are elements in an SO network.

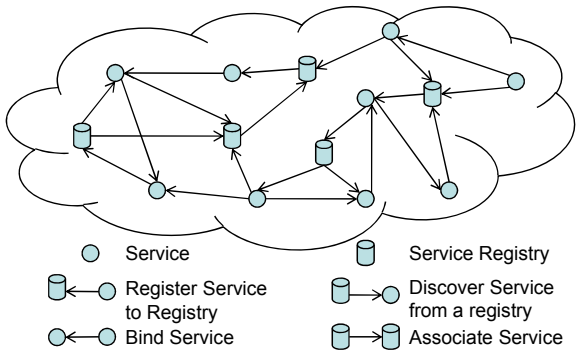

Figure 3. A network of SO elements.

When there are multiple registries in a model, one registry may also associate its registered services to other registries (which is similar to websites sharing the contents in these days). Similarly, a service may register itself to more than one registry. Figure 3 shows an example SO network.

Definition 1 (SO Network) An *SO Network* is a 3-tuple $\langle S, R, L \rangle$, where S is a set of services; R is a set of service registries; and L is a set of directed edges, each of which is a tuple $\langle e_1, e_2 \rangle$ linking e_1 to e_2 ($e_1, e_2 \in R \cup S$).

Based on the SO network, we further define the concept of a binding repository for an SO application. A binding repository is a local repository capturing the information from public service registries. It represents an SO network visible to the application. The definition is as follows:

Definition 2 (Binding Repository) For an SO network $\langle S, R, L\rangle$, a *binding repository BR* is a collection of *binding entities*. Each *binding entity* b is a 4-tuple $\langle e, I, O, PR\rangle$, where e ($\in R \cup S$) uniquely identifies b, written as $b = BR(e)$; I ($\subseteq R \cup S$) is the set of inbound inks of e; O ($\subseteq R \cup S$) is the set of outbound links of e; and PR is the probability of e (computed by some link analysis algorithm).

4. A Framework of Service Composition

In this section, we present our adaptive framework, which not only selects the most highly linked services when new services are needed, but also retains as much as possible the initial pre-deployed selection of services. To avoid being trapped at a local maximal, our framework also has a feature that selects services among those within the same tier.

In addition, when an execution of a selected service produces a failure in a service composition, the framework will remove the service from the registry. After a previously failed service is recovered, the framework computes the link analysis score of the public service registries to check how the service is being used by other consumers. Intuitively, in the long run, only "good" services will be kept in different tiers and available for selection by the consumer, and our experiment in Section 5 supports this intuition. Our framework is realized by the algorithm *COMPUTE_N_TCSM*.

Initially, information from public service registries is captured and recorded locally in the *binding repository*, and the potential services by the consumer are kept in an *N-Tier Candidate Service Module* (*N-TCSM*). Formally, an *N-TCSM* is a list of N elements, denoting N tiers. Each element is a set of candidate services (of the same or different service types). To ease our presentation, we denote the i-th tier of an *N-TCSM* by *N-TCSM*[i].

To handle a user request, the *Service Selection* component selects a set of candidate services from the *N-TCSM* so that the service consumer can form its service composition. For a service consumer who requests m services of the same kind from the *N-TCSM*, the algorithm will do the following: Starting from $i = 1$, if k ($\geq m$) such services are available in *N-TCSM*[i], then select m services randomly from these k services; otherwise, select the k services from *N-TCSM*[i], and continue to select the remaining $m - k$ services from the subsequent *N-TCSM*[i+1], *N-TCSM*[i+2], ... until a total of m services have been selected.

After executing a selected service e from *N-TCSM*[i] in a service composition, if a failure results, e will be removed from *N-TCSM*[i]. Such handling is done by the *Consumer Evaluation* component. Furthermore, the corresponding binding entity of e will also be removed from the binding repository. Whenever a service e has been removed from a tier, a new service x with the highest estimated ranking (see the *Ranking Criteria* module below) but not in the *N-TCSM* will be placed initially into the lowest tier of the *N-TCSM* (that is, *N-TCSM*[N] is updated to *N-TCSM*[N] $\cup$ $\{x\}$). The actual tier of the newly added service will be determined by the *Ranking Criteria*.

The *Ranking Criteria* component provides the ranking facility by applying the link analysis algorithm to the *binding repository*. Candidate services in the *N-TCSM* will change their belonging tiers according to the relative scores of the services computed by the link analysis algorithm. It should fully fill *N-TCSM*[i] with services with the highest scores, followed by filling *N-TCSM*[i+1] with the remaining services.

Algorithm *COMPUTE_N_TCSM*

Inputs Service consumer c, binding repository BR, SO network $\langle S, R, L\rangle$, *N-TCSM*, link analysis algorithm F

Outputs Binding repository BR, SO network $\langle S, R, L\rangle$, *N-TCSM*

// Service Selection
1 Randomly select service p from *N-TCSM*[i], $i = 1..N$.
// Consumer Evaluation
2 Collect the invocation result of p as r.
3 **if** r is correct, **then** return.
// if r is not correct, i.e., there is a failure when executing p
// Update *N-TCSM*
4 $p_{add} \leftarrow \{p_{add} \in S \mid (\forall p' \in S, BR(p_{add}).PR \geq BR(p').PR) \wedge (p_{add}, p' \notin N\text{-}TCSM[i], i = 1..N)\}$.
5 *N-TCSM* [i] $\leftarrow$ *N-TCSM* [i] $\setminus$ $\{p\}$.
6 *N-TCSM* [N] $\leftarrow$ *N-TCSM* [N] $\cup$ $\{p_{add}\}$.
// Binding Repository: Update *BR* and *SO Network*
7 Let $BR(c)$, the *binding entity* of c in BR, be $\langle c, I_c, O_c, PR_c\rangle$.
8 $O_c \leftarrow O_c \setminus \{p\}$. // Remove p
9 $O_c \leftarrow O_c \cup \{p_{add}\}$. // Add p_{add}
10 Let $BR(p)$, the *binding entity* of p in BR, be $\langle p, I_p, O_p, PR_p\rangle$.
11 $I_p \leftarrow I_p \setminus \{c\}$.
12 Let $BR(p_{add})$, the *binding entity* of p_{add} in BR, be $\langle p_{add}, I_{padd}, O_{padd}, PR_{padd}\rangle$.
13 $I_{padd} \leftarrow I_{padd} \cup \{c\}$.
14 $L \leftarrow L \setminus \{\langle c, p\rangle\}$.
15 $L \leftarrow L \cup \{\langle c, p_{add}\rangle\}$.
// Ranking Criteria: Update *PR* of services in *BR*
16 Apply F to *SO Network* to calculate the *PR* of each element e in the network, and update the *PR* of e in *BR*.
17 Sort services in *N-TCSM*[i] by their *PR* values, $i = 1..N$.

Our framework, formed by the five components above, is depicted in Figure 4.

Let us use a scenario in the *TripHandling* application to further illustrate the algorithm *COMPUTE_N_TCSM*. Suppose n candidate hotel services are bound to the workflow step *HotelBooking*, as shown in Figure 5(a). We can apply the algorithm to compute their service ranking scores. Based on the scores, the *Ranking Criteria* will rearrange the services in different tiers in the *N-Tier Candidate Service* accordingly.

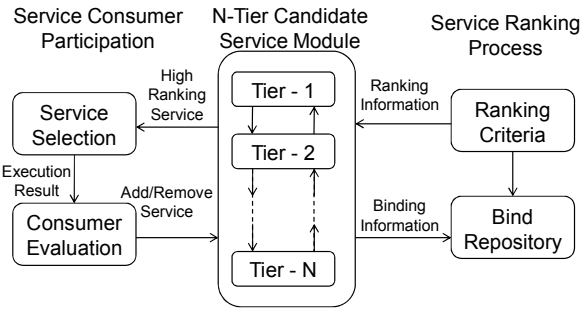

Figure 4. A framework for service selection.

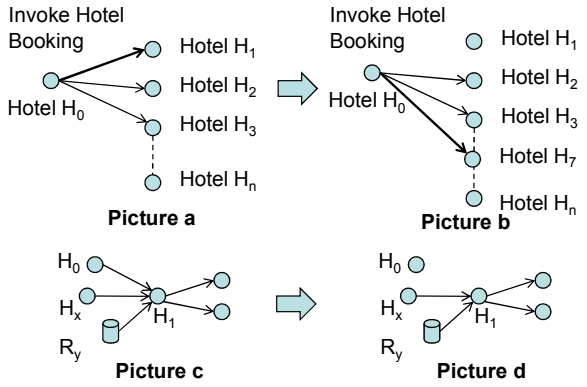

Figure 5. Example to illustrate the algorithm.

If no service in any tier is removed, the re-evaluation of the ranking scores will only rearrange the order of the services in different tiers. If some service is removed, however, the algorithm will invoke the appropriate action. Let H_0 be the hotel booking service for the *TripHandling* process, while in fact, H_0 invokes external services to do the actual booking. Suppose service H_1 (among compatible services H_1, H_2, …, H_n as shown in Figure 5(a)) is selected and executed. Unfortunately, this particular execution of H_1 results in a failure, and hence H_1 is removed from the *N-TCSM* of H_0. The link $\langle H_0, H_1 \rangle$ shown in bold in the figure is also removed from the SO network. Then, the algorithm selects a new hotel booking service (say, H_7) with a high estimated ranking to replace H_1, as shown in Figure 5(b), and the link $\langle H_0, H_7 \rangle$ is added into the SO network. During this process, the binding information for H_1 is changed, as illustrated by the transition from Figure 5(c) to Figure 5(d).

5. Evaluation

This section reports on the experimentation of our proposal.

5.1. Experiment Design

We use the *TripHandling* application [21] to evaluate our work. We have implemented a tool to automate the simulation for evaluation. It generates, in total, 200 hotel service consumers, 1000 hotel service providers, and 100 service registries. The hotel service providers are evenly distributed in four categories as shown in Table 1, in which they are classified according to the order of their failure rates. The number of service providers and the order of their failure rates for each category are shown in columns 2 and 3. Each service has a uniform distribution of failure-causing inputs, which means that any input to a service is equally likely to cause a failure.

Table 1. Failure rate settings of service providers.

Category	Count	Order of Failure Rate
L1	250	0.01%
L2	250	0.1%
L3	250	1%
L4	250	10%

We use a 3-*Tier Candidate Service Module* (or 3-*TCSM* for short) in the simulation, and label the three tiers as *reliable* tier, *available* tier, and *backup* tier, respectively. We randomly select 0.5%, 1%, and 2% of 1000 hotel services and put them into the service pools of the *reliable*, *available*, and *backup* tiers, respectively. The different settings of failure rates reflect various implementations of services.

After an invocation of a service, if it encounters a failure, the service is removed from the 3-*TCSM*. If a service has been removed from the 3-*TCSM*, a new service from the remaining service provider pool will be selected and added to the 3-*TCSM* according to our framework by applying the algorithm *COMPUTE_N_TCSM*. Since their priorities may be changed, the candidate services in the 3-*TCSM* are re-divided into different tiers according to the new ranking information.

Experiment A. Initially, each service consumer randomly selects a set of services (irrespective of the category) to compose its required service. This experimental setting simulates a random sample of pre-deployment information for initial service selection. The initial settings of all service consumers are listed in Table 2, in which columns L1 to L4 individually present the number of services in each category used by all service consumers. The maximum, average, and minimum failure rates of all selected candidate services are 10%, 2.77%, and 0.01%, respectively. Note that the total counts in columns L1 to L4 of Table 2 are not the same as the counts in the corresponding rows of Table 1, because multiple consumers may use the same candidate service.

Table 2. Initial statistics of 3-*TCSM* (metric *M-Q*).

Tier	L1	L2	L3	L4	
Tier-1 (Reliable)	543	544	562	551	
Tier-2 (Available)	602	609	601	588	**Overall Quality *M-Q***
Tier-3 (Backup)	625	580	592	603	
Total Count	1770	1733	1755	1742	
Failure Rate	0.01%	0.1%	1%	10%	**2.77%**

We choose the following two metrics as the effectiveness measures to evaluate our approach.

- The overall quality of the 3-*TCSM* (denoted by *M-Q*) is defined as the average failure rate of all candidate services in the 3-*TCSM*.
- The average failure rate per service invocation (denoted by *M-FR*) is defined as the average failure rate experienced by a consumer over a sequence of user requests for that service composition since the first request. For instance, if 5 failures result from a service composition when fulfilling a sequence of 100 user requests, then *M-FR* is 0.05.

Since PageRank is highly representative, we choose it as the link analysis algorithm *F* as input to our algorithm *COMPUTE_N_TCSM* in the experiment. Referring to [5][20], we terminate the execution of PageRank after 200 iterations. We invoke *COMPUTE_N_TCSM* 4096 (= 2^{12}) times, and repeat the experiment 10 times to report the average result.

Experiment B. We also evaluate our framework on handling services whose quality has evolved. After invoking *COMPUTE_N_TCSM* 512 (= 2^9) times, we randomly choose 30% of the services in *3-TCSM* and change their failure rates to a randomly selected rate in {0.01%, 0. 1%, 1%, 10%}. We then continue until the total number of invocations of *COMPUTE_N_TCSM* is 4096, so that we can compare the results with those obtained from Experiment A.

This change of failure rate simulates a scenario of evolving service quality. Thus, the experiment helps verify the ability of our approach to handle the evolving quality problem.

5.2. Data Analysis

We firstly report the result of Experiment A. After the simulation, the statistics of services as kept in the 3-*TCSM* is shown in Table 3. The overall quality *M-Q* after adopting our approach is 1.92%. Compared with the *M-Q* value of 2.77% for the initial setting in Table 2, the overall quality of the 3-*TCSM* services has improved by 31%.

Table 3. Average statistics of 3-*TCSM* after running simulation (metric *M-Q*).

Tier	L1	L2	L3	L4	
Tier-1 (Reliable)	1156	852	142	50	
Tier-2 (Available)	614	617	629	541	**Overall Quality *M-Q***
Tier-3 (Backup)	625	584	596	594	
Total	2395	2053	1367	1185	
Failure Rate	0.01%	0.1%	1%	10%	**1.92%**

We observe from Tables 2 and 3 that, using our approach, the chance of selecting a high-quality candidate service (say, from categories L1 and L2) has increased significantly.

We sample the 4096 invocations of our algorithm at 2^i steps, where *i* ranges from 0 to 12. The sampling results of our approach and the random approach are plotted as two data series in Figure 6. The *X*-axis indicates the number of service invocations, and the *Y*-axis indicates the failure rate.

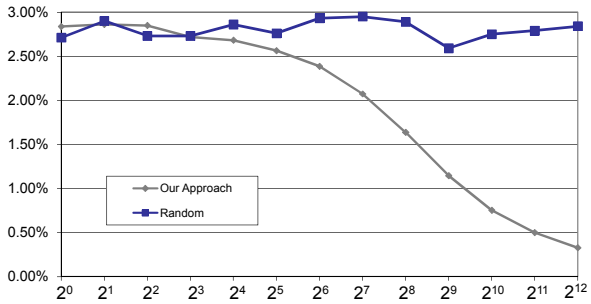

Figure 6. Comparison with random (static) (metric *M-FR*) [*X*-axis: no. of invocations; *Y*-axis: failure rate].

We observe from Figure 6 that, as the number of service invocations increases to 2^{12}, the average failure rate of executed services (*M-FR*), randomly selected from Tier-1, is 0.326%, while that of the random approach is 2.84%. Thus, the average failure rate using our approach is only 11.5% of that using the random approach.

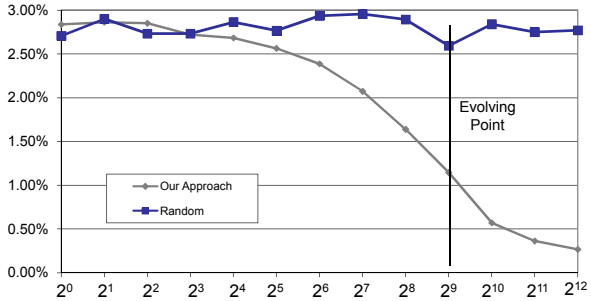

Figure 7. Comparison with random (evolving at 2^9) (metric *M-FR*) [*X*-axis: no. of invocations; *Y*-axis: failure rate].

Figure 7 shows a comparison with the random approach under the scenario of evolving service quality for Experiment B. The *X*-axis and *Y*-axis are the same as those of Figure 6. By comparing Figures 6 and 7, we observe that our approach can work even better after the quality evolving, such as achieving a failure rate of 0.265% after 2^{12} invocations in Figure 7 as against a failure rate of 0.326% in Figure 6. This observation shows that our approach is promising in solving the problem of evolving service quality. Owing to space limitation, we have to leave other details of the experiment and analysis to future publications.

5.3. Threats to Validity

In this section, we discuss the threats to validity on the design and results of our experiment.

Firstly, we use simulation to evaluate our proposal. Simulations have been used in many engineering disciplines to compare different techniques, in both research and practice. We have tried our best to model the major factors that may affect the experimental results in the experiment design, such as the failure rates of service providers and the random selection of initial service providers.

Next, we set up the initial scenarios as randomly as possible to avoid bias. The random setting may be good for comparing techniques but may not represent a general setting of the Internet. In addition, we set the number of tiers to 3, but other numbers may be used for various reasons. We tend to believe that even when the number of tiers is changed, our approach will still show similar advantages over the random technique.

We only compare our approach with a baseline (the random) technique in our experiments, which show that our approach can effectively alleviate the evolving quality problem. Comparison with other techniques will not invalidate this reasoning. The failure rates of services are changed manually in the current experiments. We plan to further automate the experimental process, thereby allowing us to simulate more evolving scenarios to evaluate our approach.

Finally, in the experiment, we assume that the environment does not affect the failure rates of candidate services. The study of context dependencies of services is a question that we shall investigate in the future.

6. Related Work

In this section, we review the literature on service composition. Since service ranking and selection approaches have been reviewed in Section 1, we do not repeat them here.

Firstly, we review context-aware service composition [8][15][18] in general. Mokhtar et al. [18] discuss the service composition problem in a pervasive computing environment. They use Ontology Web Language for Services (OWL-S), which is considered quite a complete framework to describe semantic web services and to model contexts for user tasks. Lee et al. [15] propose to apply dynamic service composition to alleviate the diversity and unpredictability problems in the context of mobile network environments. They use the smart space middleware architecture to hide the complexity in context-aware service composition.

These approaches show that, in the context of dynamic environment, the qualities of services can be changing. Hence, selecting proper services is crucial for service compositions.

Lu et al. [16] studies the composition problem in the context of workflow semantics. Their work allows developers to define workflow specifications and then to reason whether the implementations meet the specifications semantically. Our technique focuses on how clients treat their service partners expressed in links. Intuitively, the links express the tendency to form a service composition among service partners. Our technique is *softer* than the work discussed above.

Next, we outline the efforts in testing service composition. Basic techniques to test service composition are summarized in [2][7]. Bucchiarone et al. [7] introduce the testing techniques from two aspects: testing orchestrations and testing choreography. They also discuss the application of classical techniques to unit and integration testing. Members of our research group also develop service testing techniques [10][17]. Zhang et al. [24] proposes to use a Petri-net based specification model for web services to facilitate verification and monitoring of web service integration. Zhu [25] outlines a framework for testing web services. An ontology tool for software testing, STOW, is used to specify the semantics of services. In STOW, a service must provide the identity of the service provider and its capability to perform testing tasks. We, however, use the standard registries that store service binding information as one of the foundations of our work.

In the area of service selection from multiple service providers, existing projects [8][23] are in line with our work.

Zeng et al. [23] propose a middleware platform to address the issue of selecting web services, aiming at maximizing user satisfaction. They discuss service selection from two aspects: task-level selection and global allocation of tasks to services. Our approach discusses the selection of services according to their binding information, which reflects the quality of service in a statistical way. Our framework can be extended to support other evaluation criteria (such as execution price and execution duration) by adding new repositories for service ranking.

Casati et al. [8] use composite e-services to handle the dynamic environment. They propose to use service selection rules in specifications to guide service selection. Their model enhances service modeling from the specification perspective. Our work provides a more precise analysis of service ranking. We use dynamic service binding information rather than the specification.

7. Conclusion

Service ranking and selection are crucial in building composite services. Because of dynamic environmental contexts and evolving implementations of services, the quality and functionality of a service may change over time. When an organization develops an SO application, in which some of the services are provided by external partners, the organization should address these issues.

In this paper, we have proposed an adaptive framework that aims to provide better quality for the resultant service compositions. In the framework, estimated failure rates of services are initially used to rank the services into different tiers. The binding information in local service registries is updated dynamically according to the evaluation results supplied by service consumers. We then use such binding information to approximate a network of services, and apply link analysis to prioritize services in the service pool according to by their current popularity. As such, a guided service composition can be formed by using the highly ranked services. To evaluate the quality of the resultant service composition, we have conducted a simulation experiment. The experimental results show that, in terms of the number of failures experienced by consumers, our proposal significantly outperforms random selection of applicable services with or without taking evolving quality into consideration.

In the future, we plan to conduct more experimentation, and evaluate our framework using emergent properties. We shall study context dependencies and explore possible enhancements of our framework. Link analysis on the binding repository can be costly if the network is huge. It would be interesting to explore other approximation approaches, such as using link analysis on a partial service-orientation network. It will also be interesting to know the effect of applying other social network analysis approaches on top of our framework.

References

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

[2] B. Benatallah, R.M. Dijkman, M. Dumas, and Z. Maamer. Service-composition: concepts, techniques, tools and trends. In *Service-Oriented Software System Engineering: Challenges and Practices*, Z. Stojanovic and A. Dahanayake, editors, pages 48–66. Idea Group Publishing, Hershey, PA, 2005.

[3] A. Borodin, G.O. Roberts, J.S. Rosenthal, and P. Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM Transactions on Internet Technology* 5 (1): 231–297, 2005.

[4] A. Bottaro and R.S. Hall. Dynamic contextual service ranking. In *Software Composition*, volume 4829 of Lecture Notes in Computer Science, pages 129–143. Springer, Berlin, Germany, 2007.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30 (1–7): 107–117, 1998.

[6] M. Broy, I. H. Krüger, and M. Meisinger. A formal model of services. *ACM Transactions on Software Engineering and Methodology*, 16 (1): Article No. 5, 2007.

[7] A. Bucchiarone, H. Melgratti, and F. Severoni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering* (*ASSE 2007*). Mar del Plata, Argentina, 2007.

[8] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.C. Shan. Adaptive and dynamic service composition in eFlow. In *Advanced Information Systems Engineering*, volume 1789 of Lecture Notes in Computer Science, pages 13–31. Springer, Berlin, Germany, 2000.

[9] H. Cervantes and R.S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component Model. In *Proceedings of the 26th International Conference on Software Engineering* (*ICSE 2004*), pages 614–623. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[10] W.K. Chan, S.C. Cheung, and K.R.P.H. Leung. A metamorphic testing approach for online testing of service-oriented software applications. *International Journal of Web Services Research* 4(2):60–80, 2007.

[11] H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *IEEE Transactions on Knowledge and Data Engineering*, 19 (10): 1308–1320, 2007.

[12] A. Erradi and P. Maheshwari. AdaptiveBPEL: a policy-driven middleware for flexible Web services composition. In *Proceedings of the Middleware for Web Services Workshop* (*MWS 2005*), pages 5–12. Enschede, The Netherlands, 2005.

[13] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web service compositions. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering* (*ASE 2003*), pages 152–161. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[14] J. Gekas and M. Fasli. Automatic Web service composition based on graph network analysis metrics. In *On the Move to Meaningful Internet Systems 2005: CoopIS*, *DOA*, *and ODBASE*, volume 3761 of Lecture Notes in Computer Science, pages 1571–1587. Springer, Berlin, Germany, 2005.

[15] C. Lee, S. Ko, S. Lee, W. Lee, and S. Helal. Context-aware service composition for mobile network environments. In *Ubiquitous Intelligence and Computing*, volume 4611 of Lecture Notes in Computer Science, pages 941–952. Springer, Berlin, Germany, 2007.

[16] S. Lu, A. Bernstein, and P. Lewis. Automatic workflow verification and generation. *Theoretical Computer Science*, 353 (1): 71–92, 2006.

[17] L. Mei, W.K. Chan, and T.H. Tse. Data flow testing of service-oriented workflow applications. In *Proceedings of the 30th International Conference on Software Engineering* (*ICSE 2008*), pages 371-380. ACM Press, New York, NY, 2008.

[18] S.B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In *Rapid Integration of Software Engineering Techniques*, volume 3943 of Lecture Notes in Computer Science, pages 129–144. Springer, Berlin, Germany, 2006.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the Web. Stanford InfoLab Publication 1999-66. Stanford University, Palo Alto, CA, 1999. Available at http://dbpubs.stanford.edu/pub/1999-66 .

[20] M. Sobek. A survey of Google's PageRank. eFactory GmbH and Co., Dusseldorf, Germany, 2002/03. Available at http://pr.efactory.de/ .

[21] Travel handling. IBM BPEL Repository. Available at http://www.alphaworks.ibm.com/tech/bpelrepository .

[22] W.T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, and R. Paul. Testing Web services using progressive group testing. In *Content Computing*, volume 3309 of Lecture Notes in Computer Science, pages 314–322. Springer, Berlin, Germany, 2004.

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

[24] J. Zhang, C.K. Chang, J.Y. Chung, and S.W. Kim. WS-Net: a Petri-net based specification model for Web services. In *Proceedings of the IEEE International Conference on Web Services* (*ICWS 2004*), pages 420–427. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[25] H. Zhu. A framework for service-oriented testing of Web services. In *Proceedings of the 30th Annual International Computer Software and Applications Conference* (*COMPSAC 2006*), volume 2, pages 145–150. IEEE Computer Society Press, Los Alamitos, CA, 2006.