Reputation-Enhanced QoS-based Web Services Discovery

Ziqiang Xu, Patrick Martin, Wendy Powley and Farhana Zulkernine
School of Computing, Queen's University, Kingston, ON, Canada K7L 3N6
E-mail: {xu, martin, wendy, farhana}@cs.queensu.ca

Abstract

With an increasing number of Web services providing similar functionalities, Quality of Service (QoS) is becoming an important criterion for selection of the best available service. Currently the problem is twofold. The Universal Description, Discovery and Integration (UDDI) registries do not have the ability to publish the QoS information, and the authenticity of the advertised QoS information available elsewhere may be questionable.

We propose a model of reputation-enhanced QoS-based Web services discovery that combines an augmented UDDI registry to publish the QoS information and a reputation manager to assign reputation scores to the services based on customer feedback of their performance. A discovery agent facilitates QoS-based service discovery using the reputation scores in a service matching, ranking and selection algorithm. The novelty of our model lies in its simplicity and in its coordination of the above mentioned components. We present experiments to evaluate the effectiveness of our approach using a prototype implementation of the model.

1. Introduction

If multiple Web services provide the same functionality, then a Quality of Service (QoS) requirement can be used as a secondary criterion for service selection. QoS is a set of non-functional attributes like service response time, throughput, reliability, and availability [12][15]. The current Universal Description, Discovery and Integration (UDDI) registries only support Web services discovery based on the functional aspects of services [12]. The problem, therefore, is firstly to accommodate the QoS information in the UDDI, and secondly to guarantee some extent of authenticity of the published QoS information. QoS information published by the service providers may not always be accurate and up-to-date.

To validate QoS promises made by providers, we propose that consumers rate the various QoS attributes of the Web services they use. These ratings are then published to provide new customers with valuable information that can be used to rank services for selection. Web service QoS reputation can be considered as an aggregation of QoS ratings for a service from consumers over a specific period of time. This provides a general estimate of the reliability of a service provider. With service reputation taken into consideration, the probability of finding the best service can be increased. However, the assumption is that the customer ratings are considered non-malicious and fairly accurate.

There are two major problems in using QoS for service discovery. First is the specification and storage of the QoS information, and second is the specification of the customer's requirements and matching these against the information available. Major efforts in this area include Web Services Level Agreements (WSLA) [5] by IBM, Web Services Policy Framework (WS-Policy) [2], and the Ontology Web Language for Services (OWL-S) [3]. Most of these efforts represent a complex framework focusing not only on QoS specifications, but on a more complete set of aspects relating to Web services. Some researchers propose other simpler models and approaches [7][10][14] for dynamic Web services discovery. However, they all struggle with the same challenges related to QoS publishing and matching.

We propose a Web services discovery model that contains an extended UDDI to accommodate the QoS information, a reputation management system to build and maintain service reputations, and a discovery agent to facilitate the service discovery. We develop a service matching, ranking and selection algorithm based on a matching algorithm proposed by Maximilien and Singh [9]. Our algorithm finds a set of services that match the consumer's requirements, ranks these services using their QoS information and reputation scores, and finally returns the top M services (M indicates the maximum number of services to be returned) based on the consumer's preferences in the service discovery request.

The goal of this research is to investigate how dynamic Web service discovery can be realized to satisfy a customer's QoS requirements using a new model that can be accommodated within the existing

basic Web service protocols. We present simulation results executed on a prototype model in our laboratory environment. The results show the effectiveness of using a reputation management system together with the QoS information published by the service providers. It further demonstrates the efficiency of using a discovery agent with service matching, ranking and selection algorithms.

The remainder of the paper is organized as follows. Section 2 outlines the related research conducted in the area of Web services discovery, QoS and reputation. Our proposed discovery model is illustrated in Section 3. Section 4 presents simulation experiments that evaluate the effectiveness of our model and the matching, ranking and selection algorithm. We conclude in Section 5 with a summary of our work and possible future research in this direction.

2. Related Work

A number of research efforts have studied either QoS-based service discovery or reputation management systems. We provide an overview of some of this work as a context for the research discussed in the remainder of the paper.

2.1. QoS and Web Services Discovery

Blum [1] proposes to extend the use of categorization technical models, (*tModels),* within the UDDI to represent different categories of information such as version and QoS information. A Web service entry in the UDDI can refer to multiple *tModels* [13] that are registered with the UDDI, which in turn can contain multiple property information. Each property is represented by a *keyedReference* [13], which is a general-purpose structure for a name-value pair in the tModel. We use this approach of using tModels to include QoS information in the UDDI.

Ran [12] proposes an extended service discovery model containing the traditional components – the Service Provider, Service Consumer and the UDDI Registry, along with a new component called a *Certifier*. The Certifier verifies the advertised QoS of a Web service before its registration. The consumer can also verify the advertised QoS with the Certifier before binding to a Web service. Although this model incorporates QoS into the UDDI, it does not integrate consumer feedback into the service discovery process.

Gouscos *et al.* [4] propose a simple approach where important Web service quality and price attributes are identified and categorized into two groups, namely static and dynamic attributes. The Price, Promised Service Response Time (SRT) and Promised Probability of Failure (PoF) are considered static in nature and can be accommodated in the UDDI registry. The actual SRT and PoF values, which are subject to dynamic updates, can be stored either in the UDDI registry or in the WSDL document, or can be inferred at run time through a proposed information broker. The advantage of this model is its low complexity and potential for straightforward implementation.

Maximilien and Singh [8] propose an agent framework and ontology for dynamic Web services selection. Service quality can be determined collaboratively by participating service consumers and agents via the agent framework.

Although these approaches tackle the issues of incorporating QoS information into the Web services discovery process, none consider feedback from consumers.

2.2. Web Services Reputation System

Majithia *et al.* [6] propose a framework for reputation-based semantic service discovery. Ratings of services in different contexts, referring to either particular application domains, or particular types of users, are collected from service consumers by a reputation management system. A coefficient (weight) is attached to each particular context. The weight of each context reflects its importance to a particular set of users. A damping function is used to model the reduction in the reputation score over time. This function, however, only considers the time at which a reputation score is computed, and ignores the time at which a service rating is made. Our framework is similar to the one proposed by Majithia *et al.* however, we employ a different damping function and we do not consider contexts for service ratings.

Wishart *et al.* [16] present *SuperstringRep*, a service discovery protocol with a built in reputation system. The reputation system collects and manages consumer ratings of a service and provides a reputation score that reflects the overall QoS to rank the services during the service discovery process. An aging factor for the reputation score is applied to each of the ratings for a service, thus newer ratings are more significant than older ones. The value of the factor is examined in the paper. Small aging factors are found to be more responsive to changes in service activity while large factors achieve relatively stable reputation scores. We designed a reputation system based on this work, however, we consider both QoS data published by the provider and the reputation scores for service discovery.

Maximilien and Singh [7] propose an approach where software agents assist in quality-based service

selection using a specialized agency to disseminate reputation and endorsement information. Reputation is built from the aggregation of consumer ratings of a service based on historic transaction records. New services with no reputation are endorsed by trustworthy service providers or consumers before their reputation is established. No details are provided as to how the reputation score of a service is computed. Our work provides the computation details of the reputation scores and accounts for the impact of reputation on service selection.

3. Reputation-Enhanced QoS-based Service Discovery

We extend the traditional Web service model consisting of a service provider, a service consumer and a UDDI to include a discovery agent and a reputation manger, and use an augmented UDDI that contains QoS information to allow QoS-based service discovery (as shown in Figure 1). The discovery agent acts as a broker between a service consumer, a UDDI registry and a reputation manager and helps to discover Web services that satisfy the consumer's functional, QoS and reputation requirements. The reputation manager collects and processes service ratings from consumers, stores service reputation scores in a Rating Database (Rating DB), and provides the scores when requested by the discovery agent.

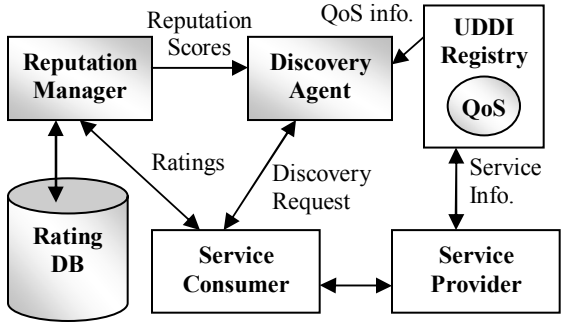

Figure 1: Model of Reputation-enhanced Web Services Discovery with QoS

3.1. UDDI Registry and QoS Information

QoS information is represented in the UDDI registry by a tModel, which is typically used to specify the technical details of a Web service. A tModel consists of a key, a name, an optional description, and a Uniform Resource Locator (URL), which points to a place where details about the actual concept represented by the tModel can be found. When a provider publishes a service in the UDDI registry, a tModel is created to represent the QoS information of the service, registered with the UDDI registry, and referenced in the *bindingTemplate* that represents the deployment information of the Web service. An Application Programming Interface (API) to the UDDI registry, such as UDDI4J [13], may be used to facilitate the operations with the UDDI. In the tModel, each QoS metric is represented by a *keyedReference,* which contains the name of a QoS attribute as *keyName,* and a *keyValue, which contains the value*.

```
<tModel tModelKey = "somecompany.com:
    StockQuoteService:PrimaryBinding:QoSInformation">
 <name>QoS Information for Stock Quote Service</name>
  <overviewDoc>
    <overviewURL>
      http://<URL describing schema of QoS attributes>
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
        tModelKey="uddi:uddi.org:QoS:Price"
        keyName="Price Per Transaction"
        keyValue=" 0.01" />
    <keyedReference
        tModelKey="uddi:uddi.org:QoS:ResponseTime"
        keyName="Average ResponseTime"
        keyValue="0.05" />
    <keyedReference
        tModelKey="uddi:uddi.org:QoS:Availability"
        keyName="Availability"
        keyValue="99.99" />
    <keyedReference
        tModelKey="uddi:uddi.org:QoS:Throughput"
        keyName=" Throughput"
        keyValue="500" />
  </categoryBag>
</tModel>
```

Figure 2: The *tModel* with the QoS information

Figure 2 shows an example of a tModel containing QoS information. The units of QoS attributes are not represented in the tModel and should ideally refer to a schema definition, which we leave to future work. For now, we assume the default units for price, response time, availability and throughput are CAN$ per transaction, seconds, percentage, and transactions per second, respectively. The example above shows the tModel for a Stock Quote service that charges CAN $0.01 per transaction, promises an average response time of 0.05 seconds, 99.99% availability, and a throughput of 500 transactions per second.

With the Web service QoS information stored in a UDDI registry, service consumers can find the services that match their QoS requirements by querying the UDDI registry. The details of this process are discussed in the following sections. A service provider should also regularly update the QoS information of

the services it publishes to ensure that the QoS information is accurate and up-to-date. To update the QoS information of a service, a service provider searches the UDDI registry to find the corresponding *tModel*, updates the QoS information in the *tModel*, and then saves it back using the same *tModelKey* that was assigned to the *tModel* when it was created.

3.2. Reputation Manager

The reputation manager collects feedback regarding the QoS of the Web services from the service consumers, calculates reputation scores, and updates these scores in the Rating DB. For this work, we assume that all ratings are available, objective and valid. Service consumers provide a rating indicating the level of satisfaction with a service after each interaction with the service. A rating is simply an integer ranging from 1 to 10, where 10 means extreme satisfaction and 1 means extreme dissatisfaction.

Our service rating storage system is similar to the one proposed by Wishart *et al.* [16]. A local database contains the reputation information which consists of service ID, consumer ID, rating value and a timestamp. The service key in the UDDI registry of the service is used as the service ID, and the IP address of the consumer is used as the consumer ID. Only the most recent rating by a customer for a service is stored in the table. New ratings from the same customers for the same service replace older ratings. The timestamp is used to determine the *aging factor* of a particular service rating.

$$U = \sum_{i=1}^{N} S_i \lambda^{d_i}$$

The reputation score (U) of a service is computed as the weighted average of all ratings the service receives from customers, where:

N is the number of ratings for the service,

S_i is the ith service rating,

λ is the inclusion factor, $0 < \lambda < 1$,

d_i is the age of the ith service rating in days.

The inclusion factor λ is used to adjust the responsiveness of the reputation score to the changes in service activity. A smaller λ means that the more recent ratings have a larger impact on the reputation score and a larger λ means more of the ratings affect the score.

3.3. Discovery Agent

A discovery agent receives service requests containing specifications for functional, QoS, and reputation requirements from the service consumer, finds the services that meet the specified criteria, and then returns a list of services to the consumer. Figure 3 shows a SOAP message for a discovery request in a general form. The strings in **bold** are replaced by the corresponding values in an actual discovery request. Generation of such SOAP messages could be automated by software, which would accept QoS as parameters and generate discovery requests as output. As shown in Figure 3, customers can specify the following in the discovery request:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <envelope xmlns =
          "http://schemas.xmlsoap.org/soap/envelope/">
   <body>
    <find_service generic="1.0" xmlns="urn:uddi-org:api">
      <functionalRequirement>
          Keywords in service name and description
      </functionalRequirement>
      <qualityRequirement weight=QoS Weight>
        <dominantQoS>Dominant QoS</dominantQoS>
        <QoS attribute 1>Value</QoS attribute 1>
        <QoS attribute 2>Value</QoS attribute 2>
        <QoS attribute 3>Value</QoS attribute 3>
        …
        <QoS attribute n>Value</ QoS attribute n>
      </qualityRequirement>
      <reputationRequirement weight=Reputation Weight>
        <reputation>Reputation Score</reputation>
      </reputationRequirement>
      <maxNumberService>Value</maxNumberService>
    </find_service>
  </body>
 </envelope>
</xml>
```

Figure 3: Service discovery request

- The maximum number of services to be returned by the discovery agent.
- Functional requirements, which are keywords in the service name and description.
- Service price is the maximum service price a customer is willing to pay.
- Service performance and other QoS requirements such as response time, throughput, and availability.
- The dominant QoS attribute.
- Service reputation requirements.
- Weights for the QoS and reputation requirements.

We assume that the same default units as described earlier for the *tModel* are used for the QoS values in the request. In future work, the units would be queried from a published schema definition and used in the query.

The *dominant QoS attribute* is the attribute deemed by the consumer to be the most important in the search criteria and is used in the calculation of the

QoS score as described later. We assume that it is easier, and more realistic, for consumers to specify one dominant QoS attribute instead of separate weights for all various QoS attributes. Average response time is considered as the default dominant QoS attribute if none is specified by the consumer. A consumer can specify only QoS requirements in the request, or both QoS and reputation requirements using separate weights for each to indicate their relative importance, where the weights for QoS and reputation requirements must sum to 1. Higher weights represent greater importance.

The calculation of QoS scores of services is performed by the equation below where $QoSScore_i$ is the QoS score of service *i, i* being the position of the service in the list of matched services, $DominantQoS_i$ is the value of the dominant QoS attribute of service *i*, *BestDominantQoS* is the highest or lowest value of the dominant QoS of the matched services when the dominant attribute is monotonically increasing or decreasing, respectively. A monotonically increasing QoS attribute means increases in the value reflects improvements in the quality, while monotonically decreasing means decreases in the value reflects improvements in the quality.

After the agent receives the discovery request, it contacts the UDDI registry to find services that match the customer's functional requirements, and retrieves their QoS information from the corresponding tModels. The agent then uses the service matching, ranking and selection algorithm described in the next section to select the top M services (M is specified by the customer in the discovery request) to return to the customer. If no service is found, the discovery agent returns an empty result to the customer.

$$QoSScore_i = \begin{cases} \dfrac{DominantQoS_i}{BestDominantQoS} & \text{--------- (1)} \\ \dfrac{BestDominantQoS}{DominantQoS_i} & \text{--------- (2)} \end{cases}$$

(1) when dominant QoS attribute is monotonically increasing
(2) when dominant QoS attribute is monotonically decreasing

3.4. Service Matching, Ranking and Selection Algorithm

Figure 4 shows a simplified version of our service selection algorithm where the leftmost numbers denote the line numbers. When the discovery agent receives a discovery request, it executes *fMatch* (line 2) which returns a list of services LS1 that meet the functional requirements. If QoS requirements are specified, *qosMatch* (line 4) is executed next on the set of services LS1 and it returns a subset of services LS2 that meet the QoS requirements. *selectServices* (line 6) always returns a list of M services to the customer where M denotes the maximum number of services to be returned as specified in the discovery request. If QoS requirements are not specified, *selectServices* returns M randomly selected services from LS1. If only one service satisfies the selection criteria, it returns this service to the customer.

```
/*Web services matching, ranking and selection algorithm */
1  findServices (functionRequirements, qosRequirements,
              repuRequirements, maxNumServices) {
     // find services that meet the functional requirements
2    fMatches = fMatch (functionRequirements);
3    if QoS requirements specified {
       // match services with QoS information
4      qMatches = qosMatch (fMatches, qosRequirements); }
5    else {
       // select max number of services to be returned
6      return selectServices (fMatches, maxNumServices,
                "random"); }
7    if reputation requirements specified {
       // matches with QoS and reputation information
8      matches = reputationRank (qMatches,
                      qosRequirements, repuRequirements);
       // select max number of services to be returned
9      return selectServices (matches, maxNumServices,
                "byQoS"); }
10   else {
       // matches with QoS information
11     matches = qosRank (qMatches, qosRequirements);
       // select max number of services to be returned
12     return selectServices (matches, maxNumServices,
                "byOverall"); }
   }
```

Figure 4: Service matching, ranking and selection algorithm

In the case where no reputation requirement is specified, *qosRank* (line 11) calculates QoS scores of the services in LS2 and returns a list of services LS3 where the services are sorted in descending order based on their QoS scores. The QoS score is calculated in the range of 0 to 1 for each service based on the dominant QoS attribute value. The service with the best dominant QoS value is assigned a score of 1. From LS3, *selectServices* (line 12) returns the top M services to the customer. If M is not specified, one service is randomly selected and returned from LS3 whose QoS score is greater than the user-specified threshold *LowLimit*. For example, if *LowLimit* is 0.9, it means all services whose QoS score is greater than 0.9 will be considered in the random selection. The random selection prevents the service with the highest QoS score from always being selected, and thus helps to balance the workload among the services that provide the same functionality and similar QoS.

In the case where a reputation requirement is specified, *reputationRank* (line 8) calculates reputation scores of the services in LS2 and returns a filtered list of services LS4 containing only those services that have a reputation score equal to or above the specified required value. Reputation scores are adjusted in the range of 0 to 1 by normalizing their reputation scores relative to the highest reputation score in the set of services as shown in the following equation. $AdjRepuScore_i$ is the adjusted reputation score of service i, i is the position of the service in the list of matched services, $RepuScore_i$ is the original reputation score of service i, and h is the highest original reputation scores of the matched services.

$$AdjRepuScore_i = \frac{RepuScore_i}{h}$$

If there is more than one service in LS4, it also calculates the QoS scores of these services as described previously. Finally, it calculates the overall scores as shown in the equation below of the services in LS4 from their corresponding QoS and reputation scores and returns a sorted list of services LS5 in descending order based on the overall score. *selectServices* (line 9) then returns a list of top M services. If M=1, one service is randomly selected from LS5 whose overall score is greater than the specified threshold *LowLimit*.

$$OverallScore_i = (QoSScore_i \times QoSWeight) + (AdjRepuScore_i \times RepuWeight)$$

In the equation, $OverallScore_i$ is the overall score of service i, where i is the position of the service in the list of matched services, $QoSScore_i$ is the QoS score of service i, *QoSWeight* is the weight of QoS requirement specified by consumers, $AdjRepuScore_i$ is the adjusted reputation score of service i, *RepuWeight* is the weight of reputation requirement specified by consumers.

4. Evaluation

This section presents experimental results to evaluate the effectiveness of our discovery algorithm. A number of programs are used to simulate various roles in the model.

- A customer simulation program generates service requests with different QoS and reputation requirements.
- A rating generator program produces new service ratings.
- A reputation manager program calculates reputation scores when requested by the discovery agent.
- A discovery agent program receives simulated requests, retrieves service QoS information, and reputation scores, if necessary, and finally runs the algorithm to select services for the consumer.

In the following experiments, we assume that all the services provide the same functionality and that every consumer request has the same functional requirements which are satisfied by all the services. We consider price, response time, availability, and throughput to be the QoS parameters and use service price to categorize services, since in most cases, customers are more sensitive to price. As the simulation progresses, new service ratings are generated, and the service reputation scores change. Experimentation showed that λ=0.75 provides relatively stable reputation scores and we will use this value in our experiments [17].

Table 1: Summary of QoS and reputation information of Services

Reputation	QoS	Price		
		Low	Intermediate	High
Poor	Low	S1	S10	S19
	Intermediate	S2	S11	S20
	High	S3	S12	S21
Acceptable	Low	S4	S13	S22
	Intermediate	S5	S14	S23
	High	S6	S15	S24
Good	Low	S7	S16	S25
	Intermediate	S8	S17	S26
	High	S9	S18	S27

4.1. Experiment 1

This experiment demonstrates that the probability of selecting a service, which best meets a customer's requirements, is improved if the customer specifies detailed QoS and reputation requirements in the discovery request. Table 1 summarizes the reputations, QoS data, and prices of 27 services (S1 - S27). A *Low* QoS value means long response time, low availability, and low throughput while *Intermediate* and *High* denote acceptable and high QoS ratings respectively. Reputation classes of *Poor*, *Acceptable* and *Good* correspond to scores of 2, 5 and 8 respectively out of 10 for example. Similarly, in our experiments price classes of *Low*, *Intermediate* and *High* correspond to costs of 0.01, 0.02 and 0.03 CAN$ per transaction, respectively.

Table 2: Summary of QoS and reputation requirements of consumers

Consumer	Requirements		
	Price (CAN$/tr)	Performance QoS (RT, AV, THR)	Reputation
C1	No	None	No
C2	0.01	None	No
C3	0.01	0.03 s, 99.95%, 700 tps	No
C4	0.01	0.03 s, 99.95%, 700 tps	8

Table 2 summarizes the QoS (RT: response time in seconds, AV: availability in percentage, THR: throughput in transactions per second, and price in CAN$ per transaction) and reputation requirements of 4 service consumers. The *dominant QoS attribute* in the QoS requirements of consumers C3 and C4 is response time. The weights for both QoS and reputation requirements are 0.5. All consumers specify that the maximum number of services to be returned is 1. C1 is only concerned about functionality, C2 and C3 have QoS preferences, and C4 has both QoS and reputation concerns for services.

For each consumer, the same service discovery request was run 50 times and the service selected for each run is shown in Figure 5. For C1, a service is randomly selected as no requirements are specified.

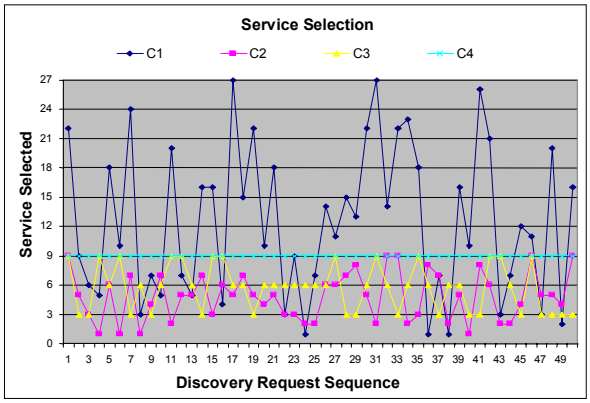

Figure 5: Experiment 1 - Service selection

For C2, a service in the low price group (S1...S9) is randomly selected. One of S3, S6 or S9 (low price, high QoS) is randomly selected for C3. S9 (low price, high QoS, good reputation) is always selected for C4.

4.2. Experiment 2

This experiment verifies that services that do not provide stable QoS performance are less likely to be selected than those which provide consistent QoS performance to customers. There are four groups of services and each group contains 4 services labeled S1, S2, S3, and S4. Table 3 shows the price and QoS advertisements for services in the four groups.

Services within the same group have different values for their actual QoS performance, and therefore, they receive different ratings from the consumers. In each group, service S1 receives average ratings from the customers during the first 10 runs of the simulation and low ratings in the next 90 runs. S2 always receives average ratings during the simulation. S3 receives average ratings during the first 10 runs and fluctuating ratings in the next 90 runs, while S4 receives average ratings during the first 10 runs of the simulation and high ratings in the next 90 runs.

Table 3: Services' price and QoS information

	Price (CAN$/tr)	QoS		
		Response Time (s)	Availability (%)	Throughput (tps)
Grp. 1	Low (0.01)	Avg. (0.05)	Avg. (99.9)	Avg. (500)
Grp. 2	High (0.03)	Short (0.02)	Avg. (99.9)	Avg. (500)
Grp. 3	High (0.03)	Avg. (0.05)	Avg. (99.9)	High (800)
Grp. 4	High (0.03)	Avg. (0.05)	High (99.99)	Avg. (500)

The QoS (including price) and reputation requirements of the four consumers (C1...C4) are summarized in Table 4. The dominant QoS attribute of consumers C3 and C4 is response time. The weights for both QoS and reputation requirements are 0.5. All consumers specify the maximum number of services to be returned as 1.

Table 4: Consumers' QoS and reputation requirements

Consumer	Requirements		
	Price (CAN$/tr)	Performance QoS (RT, AV, THR)	Reputation
C1	No	None	No
C2	0.03	None	No
C3	0.03	0.05 s, 99.9%, 500 tps	No
C4	0.03	0.05 s, 99.9%, 500 tps	8

We ran the experiment for each consumer for all 4 groups of services. For each consumer and group, the same service discovery request was run 100 times and the service selected was recorded. A service is randomly selected for customers C1, C2 and C3 from services S1, S2, S3 and S4, since all four services meet the QoS and/or reputation requirements of the three customers. S4 is selected most of the time for C4 because it provides a stable QoS performance, receives good ratings from consumers, and meets both the QoS and reputation requirements of C4. S3 is occasionally selected for C4 because it meets the QoS requirements of C4 and its fluctuating reputation score occasionally meets C4's reputation requirement. Figure 6 shows the results for consumer C4 and the services of Group 1. The results of the runs with the other groups of

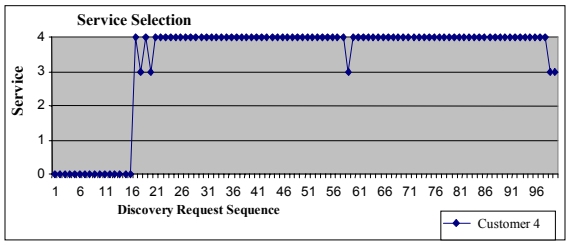

Figure 6: Experiment 2 - Service selection for consumer C4

services are similar and not shown here. Further details can be found in [17].

5. Conclusions

Due to the increasing popularity of Web service technology and the potential of dynamic service discovery and integration, multiple service providers are now providing similar services. Consumers are, therefore, concerned about the service quality in addition to the required functional properties. We propose a simple yet novel approach to provide QoS-based service discovery. Our model builds on existing Web service technology. QoS information published by the service providers in the *tModel* structure of the UDDI is used with a reputation manager to allow authentic QoS-based service discovery. A discovery agent helps finding services that meet the functional and QoS requirements specified by the consumers. With the assumption that the consumers provide non-malicious and mostly accurate QoS ratings to the reputation manager, these matched services are then ranked based on both their reputation scores generated by the reputation manager and their non-functional QoS attributes values. The top ranked services are returned to the service consumers. This way services that have high, but inaccurate, QoS values are likely to be filtered out by their low reputation scores. The paper presents an algorithm for effective service matching, ranking and selection, and demonstrates the effectiveness of the algorithm with a set of simulation experiments.

The research leads to a number of interesting avenues for future research. The model could be expanded to allow customers to specify a reputation preference. An ontology could be defined to standardize the specification of QoS attributes and their units [11]. The reliability of the reputation management system could be increased by allowing selected groups of consumers to provide the rating information or getting the raters themselves to be rated [18]. A new *stability score* may be introduced to assert the stability of the published QoS information and thus allow services that always provide good quality of service to be selected with a higher probability.

References

[1] Blum, A. (2004). "UDDI as an Extended Web Services Registry: Versioning, quality of service, and more". *White paper*, SOA World magazine, Vol. 4(6).

[2] W3C WS-Policy Framework ver.1.2 (2006). Available at: http://www.w3.org/Submission/WS-Policy/.

[3] DAML-S / OWL-S (2006). Available at: http://www.daml.org/services/owl-s/.

[4] Gouscos, D., Kalikakis, M., and Georgiadis, P. (2003). "An Approach to Modeling Web Service QoS and Provision Price". In *Proc. of the 1st Int. Web Services Quality Workshop - WQW 2003*, Rome, Italy, pp.1-10.

[5] IBM Corporation (2003). "Web Service Level Agreement (WSLA) Language Specification" Ver. 1.0. Retrieved April 30, 2006 from http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[6] Majithia, S., Shaikhali, A., Rana, O., and Walker, D. (2004). "Reputation-based Semantic Service Discovery". In *Proc. of the 13th IEEE Intl. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE), pp.297-302, Modena, Italy.

[7] Maximilien, E.M. & Singh, M.P. (2002). "Reputation and Endorsement for Web Services". *ACM SIGecom Exchanges*, Vol. 3(1), pp.24–31.

[8] Maximilien, E.M. & Singh, M.P. (2004). "A Framework and Ontology for Dynamic Web Services Selection". *IEEE Internet Computing*, Vol. 8(5), pp.84-93.

[9] Maximilien, E. and Singh, M. (2004). "Toward Autonomic Web Services Trust and Selection". In *Proc. of the 2nd Intl. conf. on Service Oriented Computing,* pp.212-221, New York City, USA.

[10] Maximilien, E. and Singh, M. (2005). "Self-Adjusting Trust and Selection for Web Services. In extended Proc. *of 2nd IEEE Intl. conf. on Autonomic Computing (ICAC)*, pp.385-386.

[11] Papaioannou, I., Tsesmetzis, D., Roussaki, I., and Anagnostou, M. (2006). "A QoS Ontology Language for Web-Services". In *Proc. of 20th Intl. conf. on Advanced Information Networking and Applications (AINA),* Vol. 1, Vienna, Austria.

[12] Ran, S. (2004). "A Model for Web Services Discovery with QoS". *SIGEcom Exchanges*, Vol. 4(1), pp.1–10.

[13] OASIS UDDI Spec TC, UDDI Ver. 2.03 "Data Structure Reference", Retrieved April 30, 2006 from http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm

[14] Vu, L., Hauswirth, M., and Aberer, K. (2005). "QoS-based service selection and ranking with trust and reputation management". In *Proc. of the Intl. conf. on Cooperative Information Systems (CoopIS),* Agia Napa, Cyprus.

[15] W3C (2003). "QoS for Web Services: Requirements and Possible Approaches". Available: http://www.w3c.or.kr/kr-office/TR/2003/NOTE-ws-qos-20031125/.

[16] Wishart, R., Robinson, R., Indulska, J., and Josang, A. (2005). "SuperstringRep: Reputation-enhanced Service Discovery". In *Proc. of the 28th Australasian conf. on Computer Science*, Vol. 38, pp.49-57.

[17] Xu, T. (2006). "Reputation-Enhanced Web service discovery with QoS", Ph.D. Dissertation, School of Computing, Queen's University, Canada.

[18] Yu, B. and Singh, M. (2002). "An evidential model of distributed reputation management". In *Proc. of the 1st Intl. joint conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna, Italy, pp.294–301.