Chapter 5

Smart Web service discovery enhanced with QoS Monitor

In Chapter 3, we have discussed two different web service registries, based on centralized approach, namely UDDI registry and ebXML registry which allow service providers to register their web services and service consumers to discover them. Though both of these registries are playing important role in e-business applications based on Service Oriented Architecture and have many things in common, in many aspects UDDI is architecturally superior to ebXML. Some more reasons to choose UDDI registry for service discovery are ebXML repositories are intended for more general purpose storage as compared to UDDI registry whereas UDDI is more focused on the kind of information that can be stored in the White, Yellow and Green pages; ebXML provide a global e-business standard of bigger size and magnitude that takes time and patience, which the industry either can't afford or chooses not to provide at this time whereas UDDI is not trying to own the world of e-business, but simply trying to facilitate all web-based services for query and introspection.

In Chapter 4, two different mechanisms based on UDDI registry are discussed in detail, which can be used to publish a web service along with its QoS information and discover a web service according the service consumer’s functional and QoS requirement. However, Reputation enhanced model for web service discovery model lacks trustworthiness of web service QoS claimed by service provider and QoS Certifier based mechanism just suggest the need of introducing a role of certifier in service oriented architecture, but lacks in implementation of the certifier. In this Chapter, we have discussed a new mechanism for web service discovery based on QoS which will rank services according to user’s preference of QoS and monitor ratings, at the same time ensuring those QoS values by monitoring them at regular intervals.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Even though many of previously discussed approaches also emphasize on web service discovery with QoS, none of them tackle the issue of trustworthiness of the published QoS information which is given by service provider themselves. Also there was no provision for specifying the priorities for each QoS parameter by the service consumer. In our proposed model, services whose QoS information is stored in the UDDI registry are monitored by Service Monitor on regular intervals and based on the deviation between published and actual QoS value, monitor ratings are given for each service. These monitor ratings are stored over the time period and used to calculate monitor score for each service. If the service consumer requests for a web service, he can specify his functional requirement, QoS requirement, domain requirement and monitor score requirement. Accordingly the Discovery agent will match, rank and select the services. Algorithms are proposed for service matching, ranking and selection which takes service monitoring into account in the ranking process.

![Service QoS Monitor Based Model For Web Services Discovery](image)

**Figure 5.1 : Service QoS Monitor Based Model For Web Services Discovery**

The standard SOA based models for web service publish and discovery comprise of three roles as service consumer who inquires for a web service, service provider who provides a web service and UDDI registry where information regarding web service is published.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

In the proposed model, apart from these three roles, the UDDI registry is enhanced with QoS information, and two additional roles as a Discovery Agent and a Service Monitor as shown in Figure 5.1. QoS information of services provided by service consumer is stored in *tModels* in UDDI registry at the time of publishing a web service. The Discovery Agent assists service consumer to find the desired web service in the UDDI registry based on his service QoS requirement, domain requirement and monitor requirement with the help of a Service Monitor. The Service Monitor regularly (every week) monitors the services for verifying the QoS information provided by the service provider at the time of service publishing and updates monitor rating database. Based on those ratings, it provides service monitor scores to the discovery agent during web service discovery.

5.1 Publishing and updating QoS Parameters

QoS information of a web service is stored in one of the data structure of UDDI registry, *tModel*. When a service provider publishes a new web service in a UDDI registry, a *tModel* is created which stores the QoS information of the service and is registered with the registry. This *tModel* is referenced in the *bindingTemplate* that represents the web service deployment. Each QoS parameter is represented by a *keyedReference* in the generated *tModel*. The QoS parameter is specified by the *keyName*, and its value is specified by the *keyValue*. We assume default units for the values of QoS parameters and hence are not represented in the *tModel*.

QoS Parameters

The international quality standard ISO 8402 (part of the ISO 9000 (ISO9000 2002)) define quality as “*the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs.*” We define quality of service as a set of non-functional characteristics that may affect the ability of the service to perform. QoS support for web services can provide a new business value to service providers by assuring a certain service quality for users. QoS parameters which we have discussed here are response time, reliability, availability, scalability and cost.

- **Response time** – The guaranteed max time required to complete a service request. In general, high quality web services should provide faster response time. The request
time is the time when the client submits a request to the Web server, and the response time is the time when the server replies after processing the request. We have considered unit for measuring response time is second. It can be measured by keeping track of the timestamps at the service request time and service response times. If the time at which the client requests for web service is $t_1$ and the time at which the client receives response is $t_2$, then the response time can be calculated as

$$\text{Response time} = t_2 - t_1$$

For Example, if the timestamp ($t_1$) of client request is 15:25:00.812 and the timestamp ($t_2$) of response to client is 15:25:01.968, then the response time of web service can be calculated as 1.156 seconds

- **Throughput** – Throughput is the number of requests completed over a period of time. Throughput can be measured by keeping track of the timestamps at the request time and response times. It is computed as the total number of requests divided by the elapsed time between the request time and the response time.

$$\text{Throughput} = \frac{\text{number of requests processed}}{\text{unit time}}$$

For Example, if the timestamp ($t_1$) of client request is 16:09:00.324 and the timestamp ($t_2$) of response to client is 16:09:00.350, then its throughput can be calculated as 1/0.026 request per second.

i.e. Throughput =~ 38 requests per second.

- **Reliability** – Reliability is the ability of a web service to perform its required functions under given conditions for a specified time interval. It also refers to the assured and ordered delivery for messages being sent and received by service requestors and service providers. Reliability determines the percentage of the times an event is completed with success. This will help service consumer to expect the probability of a failure during a transaction. Service invocation attempt may either succeed or fail, and there is no middle way in that issue. Therefore, total number of
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

service invocation attempts will be the number of failures added to the number of successful service invocations. We have measured reliability in %. Let \( d \) denote the number of days a web service is monitored for recording the number of failures. Let \( n \) be the number of failures encountered during that period. Here failures are considered as it is easy to count the number of failures than the successful service invocations. If \( N \) is the total number of events (number of successful service invocations plus number of failures), then the reliability or the success rate in one day can be derived.

\[
\text{Ratio of failure in } d \text{ days} = \frac{n}{N}
\]

\[
\text{Daily average failure rate} = \frac{n}{N \times t}
\]

\[
\text{Success rate or Reliability} = 1 - \left( \frac{n}{N \times t} \right)
\]

For Example, if the service consumer sent requests to the web server for 5 days and the number of failures was counted and the reliability was calculated as follows:

Total number of service invocation attempts (number of successful service invocation plus number of failed events) in \( d \) days (\( N \)) = 520000 where \( d = 5 \) days.

Failures in \( d \) days (\( n \)) = 20000.

Failure rate (\( n / (N \times d) \)) = 0.0077

Success rate or Reliability = 1 – 0.0077 = 0.9923

Hence, Reliability (%) = 99.23%

- Availability – Availability is the probability that the web service is up and in a readily consumable state. The web Service should be ready and available immediately when it is invoked. High availability ensures that the system failures or server failures would be less even during the peak times when there is heavy traffic to and from the server and that the given service is available continuously at all times. Let us say the “down time” is when a web service is not available. As the web service is either
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

available or unavailable, the remaining time after subtracting the down time can be termed as the “up time” that means the web service is available.

\[ \text{Availability} = 1 - \left( \frac{\text{down time}}{\text{unit time period measured}} \right) \]

If a web service \textit{ws1} is monitored over 1 week i.e. 7 days \times 24 hours \times 60 minutes \times 60 seconds = 604800 seconds, we found it was down for 2 hours i.e. 2 hours \times 60 minutes \times 60 seconds = 7200 seconds in the whole week. The availability of web service \textit{ws1} can be computed as –

\[ \text{Availability} = 1 - \left( \frac{7200}{604800} \right) = 0.9881 \]

Availability (%) of \textit{ws1} is 98.81%

- Cost – Cost is the measure of the cost of requesting a service, which is specified by service provider at the time of publishing a service. It may be charged per service requests, or could be a flat rate charged for a period of time. For Example cost of \textit{ws1} is $10 per year.

For example, a company XYZ publishes its Currency Converter service in a UDDI registry with the following QoS information:

- Response time: 0.10 second
- Throughput: 250 transaction/second
- Reliability: 99.9%
- Availability: 99.9%
- Cost: USD 0.01 per transaction
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

The company XYZ creates and registers a `tModel` that contains the QoS information for this service before it publishes the service with the UDDI registry. An Application Programming Interface (API) to the UDDI registry, such as UDDI4J [34], may be used to facilitate the service publishing process. Above Figure shows an example of this `tModel`. Service consumers can find the desired web service that match their QoS requirement with its QoS information stored in `tModels` as shown above in a UDDI registry, by querying the UDDI registry.

```
<tModel tModelKey="xyz.com:CurrencyConverterService: PrimaryBinding:QoSInformation">
  <name>QoS Information for CurrencyConverterService</name>
  <overviewDoc>
    <overviewURL>
      http://xyz.com/qos.xsd
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference>
      tModelKey="uddi:uddi.org:QoS:ResponseTime"
      keyName="ResponseTime"
      keyValue="0.10" />
    </keyedReference>
    <keyedReference>
      tModelKey="uddi:uddi.org:QoS:Throughput"
      keyName="Throughput"
      keyValue="250" />
    </keyedReference>
    <keyedReference>
      tModelKey="uddi:uddi.org:QoS:Reliability"
      keyName="Availability"
      keyValue="99.9" />
    </keyedReference>
    <keyedReference>
      tModelKey="uddi:uddi.org:QoS:Availability"
      keyName="Availability"
      keyValue="99.9" />
    </keyedReference>
    <keyedReference>
      tModelKey="uddi:uddi.org:QoS:Cost"
      keyName="Cost"
      keyValue="0.01" />
  </categoryBag>
</tModel>
```

**Figure 5.2 The tModel with the QoS information**

The company XYZ creates and registers a `tModel` that contains the QoS information for this service before it publishes the service with the UDDI registry. An Application Programming Interface (API) to the UDDI registry, such as UDDI4J [34], may be used to facilitate the service publishing process. Above Figure shows an example of this `tModel`. Service consumers can find the desired web service that match their QoS requirement with its QoS information stored in `tModels` as shown above in a UDDI registry, by querying the UDDI registry.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

If QoS information of a web service stored in tModel in UDDI registry need to be updated, only service provider has the right to modify and update it. To facilitate the process of updating QoS information, an API to the UDDI registry, such as UDDI4J may be used. At the time of updating service QoS information, first it retrieves the registered tModel for the service from the UDDI registry, updates its content and saves it with the same tModelkey.

5.1.1 Algorithm for Publishing Service in UDDI Registry

When a new service provider wants to publish his services in the UDDI registry, he has to first register with the registry. After registration a user id and password is assigned to him, with which they will create a businessEntity and save it in the registry. Under that businessEntity, a web service is published along with its QoS information according the steps shown in algorithm in Figure 5.3

<table>
<thead>
<tr>
<th>Algorithm 5.2.1: Publishing Web Service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: authToken, businessInfo, serviceInfo, QoS of service, domain</td>
</tr>
<tr>
<td><strong>Output</strong>: Published services for a.business</td>
</tr>
<tr>
<td><strong>Method</strong>:</td>
</tr>
<tr>
<td>1. Accept an authToken by passing user id and password registered at the UDDI registry</td>
</tr>
<tr>
<td>2. Create a business entity to represent the provider</td>
</tr>
<tr>
<td>3. For each service to be published</td>
</tr>
<tr>
<td>(i) Create a tModel to represent the QoS information and domain for the service, save it in the UDDI registry</td>
</tr>
<tr>
<td>(ii) Create a bindingTemplate containing a reference to the tModel</td>
</tr>
<tr>
<td>(iii) Create a service entity to represent the service that the provider is publishing</td>
</tr>
<tr>
<td>(iv) Set the reference to the bindingTemplate in the service entity</td>
</tr>
<tr>
<td>(v) Add the service entity to the business entity</td>
</tr>
<tr>
<td>4. Save the business entity in the UDDI registry, receive a businessKey and a list of service keys assigned by the UDDI registry</td>
</tr>
</tbody>
</table>

Figure 5.3 Algorithm of Publishing Web Service

Research study by Netra Patil 97
5.1.2 Algorithm for Updating Service QoS information in UDDI Registry

Over the time period, if the service provider need to update QoS information for a particular web service published in the UDD registry, he can retrieve the business entities and service entities by providing the businessKey and serviceKey. Figure 5.4 shows a detailed algorithm for service QoS information update process.

Algorithm 5.2.1: Updating Service QoS Information

Input : authToken, businessKey, serviceKey, new QoS of service

Output : Updated service with new QoS information

Method :
1. Accept an authToken by passing user id and password registered at the UDDI registry.
2. Find the business entity representing the provider with businesskey.
3. For each service to be updated
   (i) Find the service entity representing the service that is to be updated with the servicekey.
   (ii) Find and update the tModel representing the QoS information for the service with new QoS information, save it in the UDDI registry with the same tModelkey.

Figure 5.4 Algorithm of Updating Service QoS Information

5.2 Discovering Service in UDDI Registry

A service consumer sends a request for a web service to the registry in which he specifies the functional requirement, QoS and/or monitoring requirement. As a discovery agent receives the request, it matches the service requirements with the registered services, ranks the matched services according to the QoS and/or monitoring requirement specified and returns them to the consumers. Figure 5.5 shows the detailed
algorithm of how discovery agent finds services that meet a consumer’s functional, QoS and/or monitoring requirements

**Algorithm 5.2.1:** Discovering a Web Service

**Input:** authToken, functionalReq, QoSReq, domainReq

**Output:** Published services for a business

**Method:**

1. Accept an *authToken* by passing user id and password registered at the UDDI registry
2. Find services that match the customer’s functional requirements.
3. For each of the services that meet the customer’s functional requirements
   (i) Find the *service* entity representing service in the UDDI registry with the *serviceKey*.
   (vi) Find the *tModel* representing the QoS information and domain for the service.
   (vii) Add the *serviceKey* to the short listed service list if the service’s QoS information in the *tModel* meets the customer’s QoS requirements and domain match the customer’s domain specification.
4. Determine ranking of each service in the short listed list based on their QoS score and Monitor score, arrange them in ascending order of their rank and return the specified number of services based on the consumer’s requirement

**Figure 5.5 Algorithm of Discovering a Web Service**

For example, consider a company looking for a Currency Converter service for its business application. The company specify the following service requirement details as shown in Table 5.1:
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

<table>
<thead>
<tr>
<th>Service name or description : Currency Converter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QoS Parameter</strong></td>
</tr>
<tr>
<td>Response time:</td>
</tr>
<tr>
<td>Throughput:</td>
</tr>
<tr>
<td>Reliability:</td>
</tr>
<tr>
<td>Availability:</td>
</tr>
<tr>
<td>Cost:</td>
</tr>
<tr>
<td>Monitor Score:</td>
</tr>
</tbody>
</table>

Table 5.1 : Service requirements – functional, QoS and monitoring requirement

The company relies more on monitor score (more QoS assured service) than on QoS of the service, so it specifies a weight of 0.6 to the monitor requirement and a weight of 0.4 to the QoS requirement in the discovery request. Preference order for each QoS is also specified in the request as shown in Table 5.1. Figure 5.6 shows a SOAP request example for service discovery with these QoS and Monitor requirements.

```xml
<?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>
         Currency Converter
       </functionalRequirement>
       <qualityRequirement weight=0.4>
         <responseTime pref = "1">0.05</responseTime >
         <throughput pref = "4">500</throughput>
         <reliability pref = "2">99.0</reliability >
         <availability pref = "3">99.0</availability >
         <price pref = "5">0.02</price>
       </qualityRequirement>
       <monitorRequirement weight=0.6>
         <monitorScore>8</monitorScore >
       </monitorRequirement>
     </find_service>
   </body>
</envelope>
```

Figure 5.6 : Service Discovery Request SOAP Message
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 5.7: Service Discovery Response SOAP Message
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

When this SOAP request from service consumer is received by the discovery agent, it finds 10 web services that meet the requirements specified in the request, ranks them using their QoS scores and monitor scores and returns 3 top ranking services to the consumer as the request specifies the maximum number of services to be returned are 3. The discovery agent generates a SOAP message response of service discovery as shown in Figure 5.7

5.2.1 Discovery Agent Workflow

![Diagram of Discovery Agent Workflow]

Figure 5.8: Discovery Agent Workflow for Matching, Ranking and Selecting service
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 5.8 gives an idea of the high level view of the discovery agent workflow and Figure 5.9 shows the steps of an overall algorithm for discovering the services.

Algorithm 5.3.1: Overall algorithm for finding Services
Input : functReq, qosReqt, MonitorReq, maxNumServices
Output : Ranked services
Method :

```plaintext
findServices (functReq, qosReqt, domain, MonitorReq, maxNumServices)
{
    // find services that match the functional requirements
    funct_Matched = funct_Match(functReq);

    if QoS requirements are specified in request
        // find services that match QoS requirements
        qosMatches = qosMatch (funct_Matched, qosReqt);
    else
        // return services according to the maxNumServices to be returned
        return selectServices(funct_Matched, maxNumServices, “RANDOM”);

    if MonitorReq specified
        // rank matched services with QoS and Monitor scores
        shortlisted = monitorRank(qosMatches, qosReqt, MonitorReq);
        /* return services according to the maxNumServices to be returned based on overall score */
        return selectServices(shortlisted, maxNumServices, “OVERALL_SCORE”);
    else
        // rank matched services with QoS score
        shortlisted = qosRank(qosMatches, qosReqt);
        /* return services according to the maxNumServices to be returned based on QoS score */
        return selectServices(shortlisted, maxNumServices, “QOS_SCORE”);
}
```

Figure 5.9 : Overall Algorithm for service discovery

In the above algorithm,

**funct_Match** returns a list of services that satisfy the functional requirement.

**qosMatch** returns the services that satisfy the QoS requirements.
qosRank calculates QoS scores of all the services returned by the method qosMatch and returns a list of services sorted by the QoS score in descending order.

monitorRank calculates monitor score of all the services returned by the method qosMatch, removes services whose monitor scores are below the monitor requirement, calculates overall scores for remaining services and returns a list of services sorted by the overall score in descending order.

selectServices returns a list of services according to the maximum number of services to be returned in the discovery request.

5.2.1.1 Service QoS Matching

In an algorithm given in Figure 5.10, a list of functionally matched services funct_Matched and QoS requirements qosReqt are specified by service consumer are given as input. For every services in the functionally matched service list, QoS requirements specified by the consumer are matched with the published QoS information and if matched, then those QoS matched services are returned as qos_matched.

Algorithm 5.3.2: Find services that match QoS requirements

Input : funct_Matched, qosReqt
Output : Matched services
Method :

1. Initialize qos_matched to empty list.
2. For each service s in funct_Matched service list
   (i) Obtain QoS information qosPub from UDDI registry
   (ii) If qosPub is available and match with qosReqt, then add service s in qos_matched list.
3. Return service list qos_matched.

Figure 5.10 : Service QoS Matching Algorithm
5.2.1.2 Service QoS Ranking

In an algorithm given in Figure 5.11, QoS matched services \(qos\_matched\), QoS requirement values \(qosReqt\) and preference order \(pref\_val\) of each QoS specified by service consumer are given as input. Assuming the default units for QoS parameter values, first its type is checked as whether a particular QoS is mono increasing or mono decreasing. Based on the type, QoS score is calculated for each service. In calculation of QoS score, preference order \(pref\_val\) of each QoS parameter is also considered. Then services are sorted in descending order of QoS score and these QoS ranked services are returned.

Algorithm 5.3.3: Rank matched services by QoS information

**Input:** \(qos\_matched, qosReqt, pref\_val\)

**Output:** QoS Ranked services

**Method:**

1. For each service \(s\) in \(qos\_matched\) list
   
   (i) For each QoS parameter in \(qosReqt\),
       
       a) Find the highest QoS value \(bestQoSVal\)
       
       b) If \(qosReqt.QoS.type\) is monoIncreasing, then calculate \(QoS\_Score\) of each service as
       
       \[
       s.QoS\_Score = \text{sum}(qosReqt.QoS.value / bestQoSVal) \times pref\_val
       \]
       
       Else
       
       \[
       s.QoS\_Score = \text{sum}(bestQoSVal / qosReqt.QoS.value) \times pref\_val
       \]
   
2. Sort the service list on the basis of calculated \(QoS\_Score\) in descending order.
3. Return the sorted service list.

Figure 5.11 : Service QoS Ranking algorithm
5.2.2 Monitoring Service QoS

Web Services are designed, developed and used like any other typical software. However, various roles are involved during each phase of life cycle of a web service. In a web service publishing/registration phase, service provider publishes service description as well as QoS information in UDDI registry. A service consumer finds the desired services by specifying functionality i.e. keyword and preferences of QoS parameters retrieved from the UDDI registry. During this process, a service consumer also expects service quality assurance. QoS Monitor component of Smart WebService Discovery system continuously monitors web service qualities and based on the deviation found between monitored and published QoS information, gives monitor ratings to each registered service. These monitor ratings are used for calculating monitor score needed to rank the services during service discovery process.

If the service consumer specifies a monitoring requirement in the service discovery request, the discovery agent removes those services from the matched service list whose monitor score is either unavailable or below the specified requirement. During this if only one service remains, without processing further, it is returned to the consumer as it is the only service that meets the consumer’s QoS and monitor requirement. If there are more that one services meeting the consumer’s QoS and monitor requirement, QoS scores are calculated for each as described in earlier section. Monitor scores of those matched services are then adjusted using a factor $f$ so that adjusted monitor scores range from 0.1 to 1. Factor $f$ is calculated as $f = 1 / h$, where $h$ is the highest monitor score in the matched service list. Then all monitor scores are multiplied by the factor $f$ so that the the score of the service with best monitoring result is adjusted to 1, and the other services’ scores are adjusted based on their original monitor scores. At the end, the discovery agent calculates an overall score, which is as weighted sum of the QoS score and the adjusted monitor score, for all services based on the weights of the QoS and monitor requirements specified by the customer in the discovery request. Then a number of services are selected according to the maximum number of services($N$) to be returned in the request. If $N$ is greater than 1, the top $N$ services with the highest overall scores are returned to
the consumer else one service is randomly selected from those whose overall score is greater than LowLimit.

Figure 5.12 shows the steps for finding out those services from the QoS matched service list, whose monitor rating is available and meet the consumer monitor requirement.

---

**Algorithm 5.3.4**: Find QoS matched services those also match Monitor requirements

**Input**: qos_matched, qosReq, MonitorReq

**Output**: Monitor Ranked services

**Method**:

1. For each service $s$ in qos_matched list
   (i) Obtain $Monitor\_rating$ from Monitor
   (ii) If $Monitor\_rating$ is available and above $Monitor\_Req$ value, then add service $s$ in monitor_matched list
   (iii) Else remove service $s$ from the list.

2. Return the service list monitor_matched list.

---

Monitor ratings are calculated according the algorithm given in Figure 5.13. This algorithm will be auto executed weekly for monitoring the services registered in UDDI. A test client application for measuring real time QoS will be invoking all the registered web services on every week. The result of it containing monitored values of QoS along with the timestamp on which the monitoring was done is recorded in Monitor Rating database. Based on the deviations found between published QoS values and monitored QoS values during monitoring services, monitor ratings will be calculated and stored in the database which will be used for calculating monitor score of services.
Algorithm 5.3.5: Calculate Monitor rating for each service

**Input :** services, qosPub, qosMon  
**Output :** Services with Monitor rating for each service  
**Method :**

1. For each service $s$ in services list
   (i) for each qos parameter, compute $qosMon$ as
      a) for each timestamp $i$ from 1 to $n$ (no. of timestamps ie. weeks)
         \[
         qosMon = \frac{\text{sum_qos}}{n}
         \]
      b) Compute deviation factor as,
         \[
         \Delta qos\_diff = qosPub - qosMon
         \]
      c) If $\Delta qos\_diff \leq 5$ then
         rating = 10
         Else if $\Delta qos\_diff \leq 10$ then
         rating = 9
         Else if $\Delta qos\_diff \leq 15$ then
         rating = 8
         Else if $\Delta qos\_diff \leq 20$ then
         rating = 7
         Else if $\Delta qos\_diff \leq 25$ then
         rating = 6
         Else if $\Delta qos\_diff \leq 30$ then
         rating = 5
         Else if $\Delta qos\_diff \leq 35$ then
         rating = 4
         Else if $\Delta qos\_diff \leq 40$ then
         rating = 3
         Else if $\Delta qos\_diff \leq 45$ then
         rating = 2
         Else if $\Delta qos\_diff \leq 50$ then
         rating = 1
         Else
         rating = 0
   (ii) Compute final Monitor rating as,
         \[
         \text{Monitor\_rating} = \frac{\text{sum of all ratings}}{\text{number of qos parameters}}
         \]
2. Return the service list services with Monitor rating.

Figure 5.13 : Monitor Rating Algorithm
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

In this algorithm, the monitor score of a service is computed as the weighted average of all monitor ratings the service received from service monitor, where an inclusion factor is the weight attached to each of the ratings for the service. The inclusion factor $\lambda (0 < \lambda < 1)$ is used to adjust the responsiveness of the monitor score to the changes in service activity. When $\lambda$ is set to 0, all ratings, except the ones that are provided by monitor on the same day as the monitor score is computed, have a weight of 0 and are not be included in the computation. When $\lambda$ is set to 1, all ratings have equal weight of 1 and used in the computation. A smaller $\lambda$ means only recent ratings are included and a larger $\lambda$ means more ratings are included. Figure 5.14 shows an algorithm to calculate monitor score for all services and returns those services for further processing.

**Algorithm 5.3.6: Calculate Monitor score for each service**

**Input :** services, $S$ (Service Rating), Rating Time

**Output :** Services with monitor score

**Method :**

1. For each service $s$ in services list
   
   (i) Initialize $sum\_ratings$, $sum\_aging$ to 0.
   
   (ii) For each rating $i$ to $n$
       
       a) Calculate aging factor as
       
       $$aging factor (\gamma_i) = \lambda^{d_i}$$
       
       where $d_i$ = the number of the days between the current time when the monitor score is computed and the time of the $i^{th}$ rating for the service
       
       b) $sum\_ratings = sum\_ratings + S_i \times \gamma_i$
       
       c) $sum\_aging = sum\_aging + \gamma_i$
       
   (iii) Compute $monitor\_Score$ as
       
       $$monitor\_Score = \frac{sum\_ratings}{sum\_aging}$$
       
2. Return the service list services with $monitor\_Score$.

Figure 5.14 : Monitor Score Calculation Algorithm
Figure 5.15 shows the algorithm for calculating adjusted monitor score which will be used for calculating overall score of each service.

**Algorithm 5.3.7**: Calculate adjusted Monitor score for each service and rank them

**Input**: services

**Output**: Ranked service List with adjusted Monitor score for each service

**Method**:

1. Find the highest monitor score $bestMonScore$ from the service list $services$.
2. For each service $s$ in $services$ list
   
   Compute $s.adj\_monitorScore = s.monitorScore / bestMonScore$

3. Sort the service list on the basis of calculated $monitorScore$ in descending order
4. Return the service list $services$.

Figure 5.15 : Adjusted Monitor Score Calculation Algorithm

In the algorithm shown in Figure 5.16, overall score for each service is calculated with QoS Score, adjusted monitor score (obtained from above algorithms) and QoS weight, Monitor weight requirement specified in service consumer’s request.

**Algorithm 5.3.8**: Calculate overall score for each service

**Input**: services, qosWeight, monWeight

**Output**: Adjusted Monitor score for each service

**Method**: Service List with overall score for each service

1. For each service $s$ in $services$ list
   
   Compute $s.overallScore$ as
   
   $$s.overallScore = s.QoSScore \times qosWeight + s.adj\_monitorScore \times monWeight$$

2. Return the service list $services$.

Figure 5.16 : Overall Score Calculation Algorithm
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 5.17 shows algorithm for service selection according to the maximum number of services returned.

<table>
<thead>
<tr>
<th>Algorithm 5.3.9: Select services according to the maximum number of services returned</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: services, option, maxNumServices</td>
</tr>
<tr>
<td><strong>Output</strong>: Ranked services</td>
</tr>
<tr>
<td><strong>Method</strong>:</td>
</tr>
</tbody>
</table>

1. Initialize `final_servicelist` to empty list of services.
2. If `maxNumServices > 1`, then
   i. Initialize `count` to 0.
   ii. while (`count < maxNumServices` and `count < qos_matched.size()`) |
       a) `final_servicelist.add(services[count])`
       b) Increment `count` by 1.
3. Else
   i. Initialize `candidate_services` to empty list of services.
   ii. If option = “RANDOM” then,
       `candidate_services = services`
   iii. Else
       a) For each `s` in `services`
           If option = “QOS_SCORE” then
               If `s.QoS_Score >= LowLimit`, then add service `s` to `candidate_services`
           Else
               If `s.overallScore >= LowLimit`, then add service `s` to `candidate_services`
           End For
       b) `rnum = random(0, candidate_services.size())`
       c) `final_servicelist.add(services[rnum])`
4. Return `final_servicelist`.

Figure 5.17 : Service Selection Algorithm