A Framework to Support QoS-Aware Usage of Web Services*

Eunjoo Lee[1], Woosung Jung[1], Wookjin Lee[1], Youngjoo Park[1], Byungjeong Lee[2,**], Heechern Kim[3], and Chisu Wu[1]

[1] School of Computer Science and Engineering, Seoul National University, Seoul, Korea
{ejlee,wsjung,duri96,ppang,wuchisu}@selab.snu.ac.kr
[2] School of Computer Science, University of Seoul, Korea
bjlee@venus.uos.ac.kr
[3] Dept. of Computer Science, Korea National Open University, Seoul, Korea
hckim@knou.ac.kr

Abstract. Web services offer an easier, effective, efficient way to support a service paradigm. However, there is lack of adequate QoS support in using web services. In this paper, we propose a framework to support QoS aspects of web services. Three new elements are added to the original web service framework: QoS Server, QoS UDDI Broker and QoS Agent. In this framework, a requester is able to query more flexibly by using history QoS data of each web service with few load.

1 Introduction

A Web service is a software interface that describes a collection of operations that can be accessed through standardized XML messaging over the network [1] [2]. Due to the rapid growth of web service applications and the abundance of service providers, the requester must choose the "right" service provider. In such a scenario, the quality of service (QoS) serves as a benchmark to differentiate service providers [3]. However, most of today's web service implementations fail to guarantee good quality of service to their users. In fact, UDDI is currently a registry database that allows requesters to look for Web services based on their functionality and not on QoS information [4]. The lack of adequate QoS support prevents the adoption of performance sensitive web services [5].

Recently, several studies have been carried out on framework supporting web services with QoS [3], [5], [6], [7]. They all commonly dealt with QoS support framework for web services. Yet, there is lack of consideration about end users' condition. And most of the studies require some operations in requester's side, which bring about a load on end user. Typically, QoS enabled web services are associated with a service level agreement that providers present. However, there is no mechanism to quantify the trustworthiness of providers [3]. Therefore, QoS aware framework needs to be independent of provider.

In this paper, we suggest a framework to support QoS-aware usage of web services using an agent, a broker and a QoS Server. In this framework, the extensive history

* This work was supported by grant No. R01-2002-000-00135-0 from the Basic Research Program of the Korea Science & Engineering Foundation

** Corresponding author

D. Lowe and M. Gaedke (Eds.): ICWE 2005, LNCS 3579, pp. 318–327, 2005.

data of each web service can be utilized for more flexible query. History data of a web service include area, response time, access date and time, etc. A QoS Agent regularly collects that data and is consequently processed into QoS information of each web service. Each requester has diverse condition according to the requester's usage pattern. That is, a requester uses a web service only during peak hours and the requester ignores performance aspects of the web service during off-hours. Those are expressed as requester-specific conditions in a requester's query. History data is utilized to find the most suitable web service that meets the specific conditions.

The remaining parts of this paper are organized as follows. Section 2 presents the related works and the inherent problems. Section 3 presents our proposed framework. Section 4 describes a scenario illustrating an operation in a web service where our framework is applied. Section 5 presents comparisons between our framework and other frameworks. Finally, Sect. 6 describes contribution and limitation of our study.

2 Related Works

Several studies on a web service framework with QoS support have been carried out thus far. Tian et al. presented an approach that allows not only QoS integration in Web services, but also the selection of appropriate services based on QoS requirements [7]. They classified QoS parameters as 'Web service layer QoS support' and 'the network layer QoS support'. Processing time, availability and various other requirements were included in the former. Bandwidth, packet loss and several other data points were included in the latter. A Web service Broker (WSB) was used in the proposed architecture, which fetches information of offers that requester could be interested in. WSB tests the web services that meet functional requirements and it returns the most appropriate services to requester. That is, WSB helps the requester to find appropriate offers according to 'Web service layer QoS support'. The requester and server have a requester side QoS proxy and server side QoS proxy, respectively. The proxies handle 'the network layer QoS support' at runtime. Two implementations of WSB exist; one uses a local object within the application and the other uses a remote web service. In [7], there is lack of feedback mechanism that can reflect requester's state. Also, there is no way of using history data related to QoS properties.

Singhera et al. extend web services framework installed with a Monitor [6]. In this framework, non-functional characteristics of a web service are determined at run-time and requesters are offered services that best meet their non-functional requirements as well as functional requirements. Monitor consists of a Registration Module, Service Monitoring Module, Uplink Module, Rules Engine and History Analyzer. These modules collect the non-functional characteristics, perform the necessary tests and select the best web service.

In [3], architecture with verity and which uses a reputation system was introduced. The reputation system is intended to aggregate and disseminate feedback on the past behaviors of the participants [8]. This system encourages trustworthiness and help requester choose the most appropriate system for service request. A requester and a service broker contain different profiles of the reputation vector. The service broker calculates the global verity, while the requester calculates the local verity. To determine the verity value used in quality driven selection, the user employs a weighted sum mechanism. However, most requesters are reluctant to store database and a cal-

culator in their server. Moreover, they are particularly sensitive about leakage of their server logs to the outside. Another drawback is that it is difficult to manage certain elements such as a calculator in requester. Yu and Lin proposed two resource allocation algorithms (Homogeneous Resource Allocation algorithm; HQ and Non-homogeneous Resource Allocation algorithm; RQ) [5]. The purpose of the algorithms is to achieve a high average system utility and avoid making frequent resource reconfigurations. The resource allocation in a server can be effectively adjusted so that requesters can avoid experiencing unstable performance.

3 Proposed Framework

A general web service framework is composed of three parts: Provider, Requester and discovery mechanism like UDDI. Requester and Provider can communicate via message using XML, SOAP, WSDL technology. Yet, there are only few mechanisms that can reflect Requester's QoS requirements.

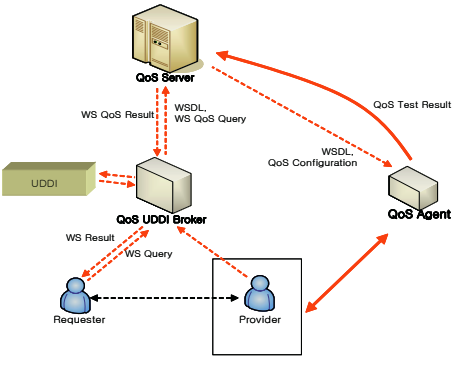


Fig. 1. QoS-Aware Framework

A QoS UDDI Broker obtains suitable set of web services, which is determined by some information in a query sent by a Requester. Figure 2 shows an example of a query in XML. The information includes not only data about general UDDI search for functionally suitable web services, but each weight value for QoS properties and condition information about time, period and area. A wide spectrum of metrics that reflect the QoS of web services have been introduced by the research community with often varying interpretation [3][9]. We consider 'reputation', 'response time', 'availability', 'reliability' as QoS properties in our example, and those QoS properties can be measured with the existing QoS metrics. QoS properties can be added to or removed from those according to a policy. QoS Server has history data for each web service that have been registered in the QoS Server. Each registered web service has *wsid*, its unique identifier.

At first, functionally suitable web services are returned to UDDI QoS Broker from UDDI. If some web services don't have *wsid*, the web services are newly registered in the QoS Server. Next, A QoS Server calculates metric value of each QoS properties based on its history data using search conditions, such as time, period and area. Finally, each weight value is applied to get the last resulting list sorted by decreasing

order according to user's preference. In addition to the list, evaluated data (grade) of each web service in the list are shown. And more, the history data is illustrated as a form of graph, which helps a user know the usage pattern of the web service and select more appropriate one. The whole selecting process is described in Fig. 3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<QoS_Query>
<Category id="1" />
<Keywords>
        <Keyword>medical</Keyword>
        <Keyword>record</Keyword>
 </Keywords>
<Weight>
          <Availability>30</Availability>
          <Reliability>40</Reliability>
          <Reputation>10</Reputation>
          <ResponseTime>20</ResponseTime>
 </Weight>
<Period>
            <FromDate>2004-02-01</FromDate>
            <ToDate>2005-01-31</ToDate>
 </Period>
  <Area id="3" />
</QoS_Query>
```

Fig. 2. An example of a query

1. S={s_1, s_2, …, s_n} // S is a set of functionally suitable web services, s_1, s_2. …, s_n.
2. Extract condition and weights for QoS properties from query.
   cond={time, area, period}
   W=<w_1, w_2, …, w_m> // W is a set of weights for each QoS properties.
3. Determine grades of each web services based on the condition using the history data. A group of web services that have no history data is listed separately.
   G_i={g_1, ..., g_m} // G_i is s_i's grade for each QoS properties
4. Calculate ranks of the web service with this following formula.
   $$f(s_i) = \sum_{k=1}^{m} w_k \bullet g_k$$ ($f(s_i)$ is an evaluation function for each web service s_i)
5. Get the network delay time between a requester and each provider of candidate web services using some technique such as '*ping*'.
6. Return the list of resulting web services with network delay time in decreasing order.

Fig. 3. Process for selecting QoS suitable web services

Compared to distributed components operated on intranet environments, web services are expected to be used based on not only intranet, but also internet. Therefore we decompose 'response time' into local response time in a provider's side and network delay time between an agent and a provider. And then, we take local response time as 'response time', for network delay may be serious factor in internet environment. Local response time can be obtained with actual response time and network delay time. In our framework, several agents test at diverse area because network status tends to be varied according to some condition, for example, area. If a web service is tested by only one agent, the result may be distorted for the condition reason.

A QoS Agent regularly tests web services registered in a QoS Server according to the configuration forwarded by a QoS Server, and it stores the test result in QoS

Server as history data of each web services. Figure 4 shows a main part of an example of QoS configuration delivered from a QoS Server to a QoS Agent. QoS configuration mainly includes testing condition (time interval, timeout). When a QoS Server is established initially, a manager registers a list of web services as one of initial setting to collect history data for the first search. When a QoS Agent tests a web service, it uses test data automatically generated from a WSDL of the web service, particularly the type information of parameters in the WSDL. When errors occur in testing due to test data, it should be fixed manually with a log message about the error.

UDDI tracks on changes on web services [10]. When a web service is changed, QoS Agent updates test modules if necessary. When a web service is not available any more, *wsid* of the web service is unregistered in the QoS Server.

```
<?xml version="1.0" encoding="euc-kr" ?>
<AgentCommands>
  <!-- From QoS Server To QoS Agent -->
  <Configuration>
      <!-- Agent Overall Configuration -->
       <Interval unit="min">10</Interval>
       <Timeout unit="sec">30</Timeout>
   </Configuration>
   <Command>
      <Configuration>
      <!-- Configuration For Target -->
       <Interval unit="min">10</Interval>
       <Timeout unit="sec">30</Timeout>
       </Configuration>
       <Target type="URI">http://violet.snu.ac.kr/wsdl/TestWebService.wsdl</Target>
      <Operations>
          <!—Operations to be tested -->
          <Operation id="/definitions/portType/operation[@name='getCharCount']">
            <!-- test configuration -->
            <Configuration>
                <!-- Configuration For Operation -->
                 <Interval unit="min">10</Interval>
                 <Timeout unit="sec">30</Timeout>
              </Configuration>
          <TestMsg>
            <!-- Test SOAP Message -->
            <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:m="some-uri">
                <soap:Body>
                  <m:getCharCountRequest>
                      <arg0>0321146182</arg0>
                   </m:getCharCountRequest>
                </soap:Body>
            </soap:Envelope>
          </TestMsg>
          <ResultMsg>
          .....
```

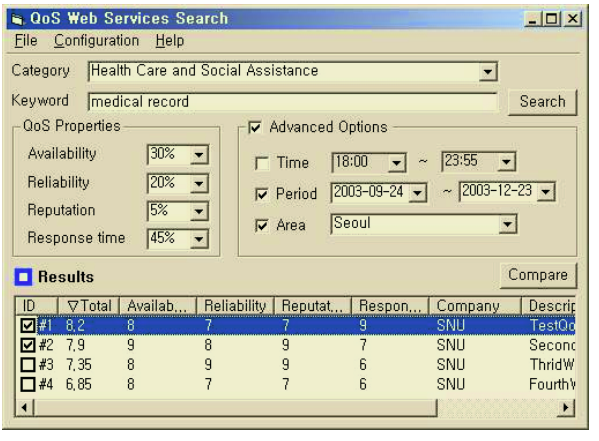
Fig. 4. An example of QoS configuration

4 Scenario Under the Proposed Framework

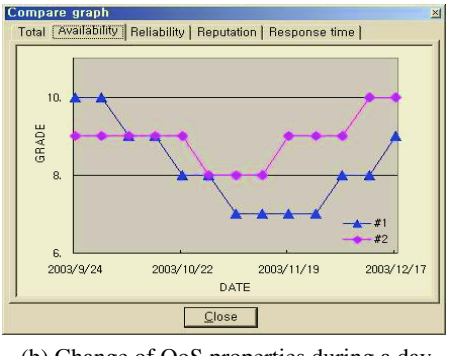
This section presents a scenario of a requester using the web services that meets both its functional and non-functional requirements by applying our framework. We also show a simple example of a conditional QoS query at the end of this section. We

assume that the various web services are defined in WSDL and registered in UDDI registries.

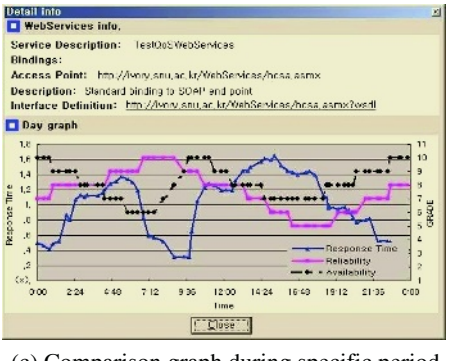
1. The QoS Server let a QoS Agent regularly test web services and collect some data such as testing date, time, area and response time, etc., which are processed into QoS information by QoS Server. QoS information of web services becomes piled up in the database. The QoS information of each web service is graded by QoS Server's grading scheme.
2. A requester sends a query to the QoS UDDI Broker. Functional requirement, QoS requirement and requester-specific condition can be included in that query. QoS requirements are a set of pairs, <QoS information, weight value>, for example, {<reliability, 35%>, <availability, 45%>, <response time, 20%>} (Fig. 5(a)). A requester can select QoS properties such as reliability, availability. If any property is not selected, all properties are considered to have same weight value.
3. A QoS UDDI Broker extracts functional parts and sends them to UDDI. UDDI finds the web services that is compatible with functional part in that query and returns the list of the web services to the QoS UDDI Broker.



(a) List of QoS suitable web services



(b) Change of QoS properties during a day



(c) Comparison graph during specific period

Fig. 5. A sample screenshot of search result

4. To the functionally suitable web services listed in the previous step 3, following process are applied to show a list of QoS suitable web services. Figure 5(a) shows the result of this search process. The functional and QoS requirements in Fig.5(a) is transformed into query as a form shown in Fig. 2.
   (a) Basic search process
   QoS Server lists resulting web services with total ranking order. Total ranking can be calculated with each weight value and predefined QoS grade.
   (b) Advanced search process
   If requester-specific condition exists in that query, the condition is applied for advanced search process. For example, if the condition is "{<area, Seoul>, <time, P.M. 8:00-P.M.11:00>}", advanced search process are executed onto web services' history data that have been recorded by a QoS Agent that had tested the web services in Seoul, from eight to eleven o'clock P.M. Total ranking is determined based on the history data in accord with the specific condition.

   Change graphs for QoS properties are displayed, which can help requester select a more suitable web service. Two types of graph are provided. One shows comparison among selected web services during specified period (Fig. 5(b)). Fig. 5(b) shows change of specific QoS property 'availability' for selected web services '#1' and '#2'. The other illustrates change of QoS properties for a specific web services during a specific day (Fig. 5(c)). This one day graph is helpful to building an application that is expected to be excessively used in specific time.
5. A requester obtains a list of web services that also include the QoS data concerned with it. A requester can choose the most suitable web service and it can be accessed with SOAP that can be automatically made from WSDL given by the provider.
6. After a requester finally selects and uses a web service, the requester can give a QoS Agent a rate of the QoS UDDI Broker for reputation properties (optional).

In this framework, QoS Agent tests and collects test results of QoS properties of each web service, in advance. Accordingly, it becomes unnecessary to check the value of QoS properties of web services at real time. Furthermore, the history data collected by QoS Agent are saved and managed by QoS Server, which can be used at various parts. In other words, requesters can deliver a QoS query more flexibly and extensively. In our framework, it is not mandatory for requesters to take part in checking reputation ranking. And at the same time, additional parts such as Local verity calculator in [3] need not be located in requesters' side.

5 Comparison

We conducted comparisons between several QoS support frameworks [3], [6], [7] with our framework. Table 1 outlines the comparison result for each comparison item. The frameworks that we considered are briefly described in Section 2. We shall refer to the framework in [3] as VERITY, that in [6] as EWSF, that in [7] as WSBF, and our framework as QAF, for convenience. Data related to QoS properties are logged in history data and they can be useful in web service selection. Usually a requester selects a web service in practice by using the results of QoS test, which in-

cludes not only the current data but also past data, for the right selection. Storage capacity has to be enough to store the massive history data. In that respect, QAF shall have the demerit of storage demand. That is, there is a tradeoff between the history data and storage demand.

In QAF, a requester is able to query more flexibly compared with others. For example, with QAF, it would be possible to execute operations as described in the previous section by using the history data. It is expected to be helpful in real usage of web service, because a requester may want to know a specific condition where web services are used. In VERITY and our QAF, requesters' feedback is supported explicitly. But VERITY allows the requesters to maintain database and a calculator in their server, and at the same time, allows their logs to be used outside. This is a burden to the requesters, which in turn poses as a usage barrier in the real market. To the contrary, QAF does not force requesters to provide feedbacks, and moreover, only the QoS Agent performs all miscellaneous operations.

The item, Requester's view, is a measure of what the web service takes into consideration as a requester. Local verity in VERITY can reflect requester's position but it has a tradeoff relationship with 'Requester's load'. In our QAF, QoS Agent executes the test of finally selected web services under requester's condition such that the condition of the requester is partly reflected on the test results.

For dynamic QoS aspect, proxies in a requester and a server handle the network layer QoS support concerning the network performance in WSBF. EWSF supports run-time non-functional QoS parameters by relying on a monitor. Similarly to EWSF, QAF makes use of history data but provides more flexible and extensive usage of the data. QAF and VERITY support users' feedback, however QAF does not put a burden on the requester with mandatory provisions of feedbacks nor require for additional elements to be installed in the requester's side as in the case of VERITY. WSBF takes into consideration dynamic aspects that are not presented to QAF. And yet WSBF doesn't utilize helpful history QoS data and feedback mechanism. Also, its local WSB is operated in the requester side.

QAF focuses on the flexibility of the usage of QoS support web service from a requester's perspective. In addition, it activates QoS Agent performing concerned executions, which lessens the load of a requester.

Table 1. Comparison with other frameworks

	VERITY	EWSF	WSBF	QAF
History data	not use	use	not use	use
Requester's view	concern	ignore	partly concern	concern
Flexibility of query	low	low	low	high
Feedback	support (mandatory)	ignore	ignore	support (voluntary)
Storage demand	medium	high	medium	high
Requester's load	heavy	light	medium	light
Dynamic QoS	ignore	partly support	support	ignore

6 Conclusion

We have proposed a QoS aware framework that can support the QoS aspects of web services. QoS Server, QoS UDDI Broker and QoS Agent have been newly added to

the original web service framework, where a requester could initially find web services to meet only functional requirements. QoS appropriate web services are selected based on QoS history data that are regularly collected by QoS Agents. In our framework, history data are utilized for more flexible and extensive usage and additional parts are not necessarily located on the requesters' side. Furthermore, the requester's view is considered and load of requester is reduced. The main contribution of our framework can be summarized as follows:

- Requesters can query more flexibly, fitting to their specific condition. Also, the resulting web services is visually shown as graph with their history data in many ways, which helps user select suitable one.
- History data can be utilized for obtaining more information and estimating the future usage of web services through data mining process.
- In our framework, selecting scheme is not dependent on provider; testing data are generated using type information in WSDL of web services, and test results are managed by a QoS Server. Therefore, requesters can get more unbiased QoS information of concerned web service.
- A requester can get tailored results using the rich set of history data and flexible query with no additional processing. Most web services did not consider a specific user's environment and preference in QoS support. Even if considering, special data such as a user's profile [3] is needed. However, we let several agents test regularly in diverse area, and we decompose response time into an agent's network delay and local response time in a provider's side. And more, requesters can actively reflect their condition and preference using extensive query.

Our framework has several limitations:

- We restrict our work to a single web service, not concerning web service composition.
- We assume the testing is conducted on mutual agreement between agents and each provider. Only authorized agent can access a provider, which requires considering security aspect.
- Dynamic QoS support is not considered. History data cannot fully satisfy the performance requirement in dynamic situation.

In the future work, we will focus on implementing three elements, QoS Server, QoS UDDI Broker, and QoS Agent. Furthermore, we'll extend our framework to achieve a more adaptive usage of web services supported by the network layer.

References

1. IBM Web services, http://www-136.ibm.com/developerworks/webservices/
2. W3C, "Web Servicesk Description Requirements." W3C Working Draft, October 2002.
3. 3. Kalepu, S., Krishnaswamy, S. and Loke, S. W., Verity: A QoS Metric for Selecting Web services and Providers, the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), pp.131 – 139, 2003.
4. UDDI, "UDDI Technical White Paper." uddi.org, September 2000.
5. Yu, T. and Lin, K., The Design of QoS Broker Algorithms for QoS-Capable Web services, IEEE Conference on e-Technology, e-Commerce and e-Service(EEE'04), 2004.

6. Singhera, Z. U., Extended Web services Framework to Meet Non-Functional Requirements, International Symposium on Applications and the Internet Workshops (SAINTW'04), 2004.
7. Tian, M., Gramm, A., Naumowicz, T., Ritter, H., and Schiller, J., A Concept for QoS Integration in Web services, the Fourth International Conference on Web Information Systems Engineering Workshops(WISEW'03), 2003.
8. Resnick, P., Zeckhauser, R., Friedman, E., and Kuwabara, K., Reputation Systems, Communications of the ACM Volume 43, Issue 12, December 2000.
9. Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M, Kalagnanam, J., and Chang, H., QoS-Aware Middleware for Web Services Composition, IEEE Transactions on Software Engineering, Vol. 30, No. 5, May 2004.
10. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm#_Toc85908401