A QoS broker based architecture for efficient web services selection

*M.Adel Serhani, *Rachida Dssouli, **Abdelhakim Hafid, **Houari Sahraoui
*Concordia University Department of Electrical and Computer Engineering 1455 de Maisonneuve Blvd. West Montreal, Quebec, H3G 1M8, Canada
**Département d'Informatique et recherche opérationnelle Université de Montréal CP 6128 succ Centre-Ville, Montréal QC H3C 3J7, Canada
{m_serhan, dssouli}@ece.concordia.ca, {ahafid, sahraouh}@iro.umontreal.ca

Abstract

Quality of Service (QoS) support in web services plays a great role for the success of this emerging technology. In this paper, we present a QoS broker-based architecture for web services. The main goal of the architecture is to support the client in selecting web services based on his/her required QoS. To achieve this goal, we propose a two-phase verification technique that is performed by a third party broker. The first phase consists of syntactic and semantic verification of the service interface description including the QoS parameters description. The second phase consists of applying a measurement technique to compute the QoS metrics stated in the service interface and compares their values with the claimed one. This is used to verify the conformity of a web service from the QoS point of view (QoS testing). A methodological approach to generate QoS test cases, as input to QoS verification is used. We have implemented a prototype that includes the verification and certification components of the broker. We performed experiments to evaluate the importance of verification and certification features in the selection process using real web services.

1. Introduction

QoS support for web services is among the hot topics attracting both researchers from academia and industry. During the emergence of web services technologies, researchers focused more on the functional and interfacing aspects of web services (i.e. Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), etc.). QoS delivered to a client may be affected by many factors, including the performance of the web service itself, the hosting platform, and the underlying network. QoS management has been extensively studied in network-based multimedia applications as well as web-based applications. In the context of web services, the research issue is very recent.

Nowadays, both Web Services providers and clients are concerned with the QoS guaranteed by web services. From the client point of view, web service based QoS selection is a multi-criteria decision mechanism that requires knowledge about the service and its QoS description. However, most of clients are not experienced enough to obtain the best selection of web service based on its described QoS. They simply trust the QoS information published by the provider; however most of web services providers do not guarantee and assure the level of QoS offered by their web services.

An open and multi-player testing environment is of paramount importance for the efficient selection of web services. This will enable third parties including web services clients and third party certification entities to verify the conformity of the features as well as the consistency of the QoS claimed by web service providers. A set of verification procedures is essential for providers to remain competitive and for clients to make the right selection and trust the published QoS metrics. Performing QoS verification is not an easy task since it is done at runtime and requires considerable information exchanges between entities involved in this process (provider, broker, and clients). Therefore, it is essential for the success of any QoS based web services architecture to support a set of novel features: (1) QoS verification and certification to guide web services selection; (2) QoS-aware web services specification, publication, and discovery; (3) QoS measurement and monitoring. In this paper, we propose a broker-based architecture for web services selection and QoS management. The role of the QoS broker within the architecture is to support QoS

Proceedings of the IEEE International Conference on Web Services (ICWS'05)
0-7695-2409-5/05 $20.00 **IEEE**

provisioning and assurance in delivering web services. It introduces and implements the concept of QoS verification and certification, which is used together with the QoS requirements in the selection process of web services. The proposed QoS broker is to be used as a third party Web Service, itself published in UDDI registries. It is invoked when a user requests a web service with QoS requirements. We present the operations of the QoS broker while processing user requests with QoS requirements.

The remainder of the paper is organized as follows. In Section 2, we briefly present related research and identify the limitations of existing approaches dealing with QoS for web services. In Section 3, we describe the architecture and the design of our proposed QoS broker. Section 4 describes the implementation of the QoS verification and certification modules Section 5 introduces the prototype implementation Section 6 concludes the paper and presents future research investigations.

2. Background and related work

Web services paradigm is a recent concept of emerging web applications. It connects a set of technologies, protocols, and languages to allow automatic communication between web applications through the Internet. A Web Service is an application that exposes its functionality through an interface description and makes it publicly available for use by other programs. As web services are a new emerging technology, most existing work focuses more on their development and their interfacing practices. QoS support in web services, and in particular QoS management, is still an immature research area. Efforts are still carried for enumerating the requirements and defining the approaches. In addition, standard web services protocols such as WSDL and UDDI were designed mainly for their functional features with only minor consideration for QoS support and verification. Until recently, considerable efforts have been conducted to work on QoS for web services. DAML-S provided an upper ontology for semantic description of web services, including specification of functionalities and QoS constraints [8]. IBM proposes Web Service Level Agreements (WSLA), which is an XML specification of SLAs for Web Services, focusing on QoS constraints [9]. Web Service Offerings Language (WSOL) has been developed for the formal specification of various constraints, management statements, and classes of service for Web Services [10]. Early framework supporting QoS-enabled web services are proposed in [8, 12]. [7] proposes a model for web services discovery that includes the functional and non-functional requirements of web services (i.e. QoS). A certification approach is introduced in the proposed framework; the goal is to certify QoS claims by providers and verify these QoS claims for the clients. The certifier introduced in the architecture [8] is not well defined and not implemented; it does not describe the details of the certification process. Furthermore, it neither verifies the WSDL content nor controls the delivery of the selected QoS. In [11], authors present a description and an implementation of broker-based architecture for controlling QoS of web services. The broker acts as an intermediary third party to make web services selection and QoS negotiation on behalf of the client. Delegation of selection and negotiation raises trustworthiness issues mainly for clients. Performance of the broker is not considered in this approach. Moreover, performance of the broker can be a key to the success of any proposed architecture; if the user does not get a response to his/her request with an acceptable response time, he/she will switch to another provider. Some similar broker based architectures were presented in [12] and [13] that focus more on the QoS specification using XML schema, and dynamic QoS mapping between server and network performance. In [14], Tsai et al suggested test scripts specification techniques to perform testing with the UDDI server. The verification tests are performed in UDDI registry that does not support QoS-aware web service publication and discovery. Most of the above works do not consider performance evaluation of web services and scalability issue while the number of clients is continuously increasing and their requirements are always changing.

In the next section, we describe the design of the proposed QoS broker-based architecture; we describe in details the QoS verification and certification functions.

3. QoS broker based architecture: components and interfaces

3.1. Architecture description

The architecture extends the standard Service Oriented Architecture (SOA) [1][2] with QoS support for web services. It includes QoS description during the service publication, and performs dynamic QoS-aware invocations. In addition, it verifies, certifies, confirms, and monitors QoS dynamically via a web service-based broker. The architecture involves four main participating roles the web service broker, the web service provider, the client, in addition to a QoS-enabled UDDIe registry [15]. Components of the architecture are presented in figure 1. A sequence of

interactions between these components is presented in figure 2.

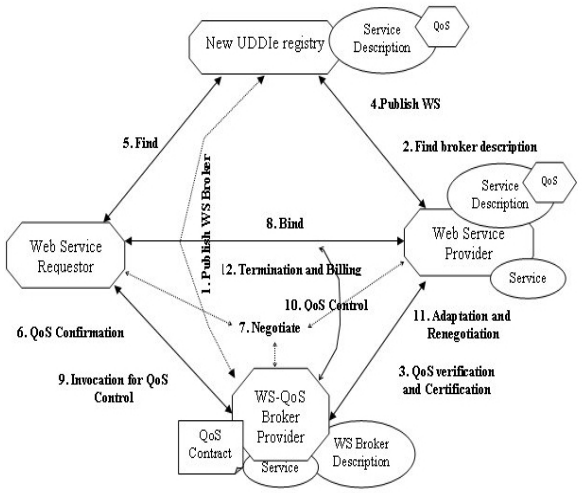

Figure 1. A QoS broker-based architecture

Figure 1 presents an architecture based broker with features that overcome limitations, of existing approaches, described above. Its important features include the support of service selection based on client requirement, QoS verification and certification. QoS verification is the process of validating the correctness of information described in the service interface as well as the described QoS parameters. The QoS verification is performed using an approach that generates test cases to measure QoS parameters. The verification will be used as input for the certification process that will be issued when the verification succeed. The broker arbitrates the negotiation process between clients and their providers until they reach an agreement. During web service invocation, the broker measures dynamically QoS attributes and uses their values to monitor the provision of the selected QoS level; then, it notifies the interested entities of any violation. The broker updates, regularly, its database whenever significant changes happen. In the architecture, the certification process goes beyond certifying just the QoS provider's claims. Additional tests can be performed to make sure that these QoS claims are fulfilled.

The broker publishes its interface description in the UDDIe registry (operation 1 in Figure 1). A web services provider looks for the broker's WSDL document in the UDDIe registry (operation 2). Then, it requests the broker to certify the web services and their supported QoS (operation 3). The certification is performed before issuing a certificate, the provider publishes his/her QoS-aware web services in the UDDIe registry (operation 4). Clients can check the UDDIe registry for QoS-enabled web services satisfying their needs (operation 5). Before starting in the negotiation process with the provider, clients have the possibility to confirm that the published classes of QoS have been previously certified by the broker (operation 6). The broker arbitrates the QoS negotiation between the client and the provider (operation 7). If an agreement is reached, the client binds to the web service using the agreed class of QoS (operation 8). During invocation, the client can ask the broker to monitor and control the delivered QoS (operation 9 and 10). If the QoS degrades, the broker notifies the provider who initiates QoS adaptation in order to maintain the agreed QoS (operation 11). The QoS renegotiation is initiated if the adaptation operations fail to maintain the agreed QoS (operation 11). The processes terminate by releasing resources and issuing the corresponding bill (operation 12).

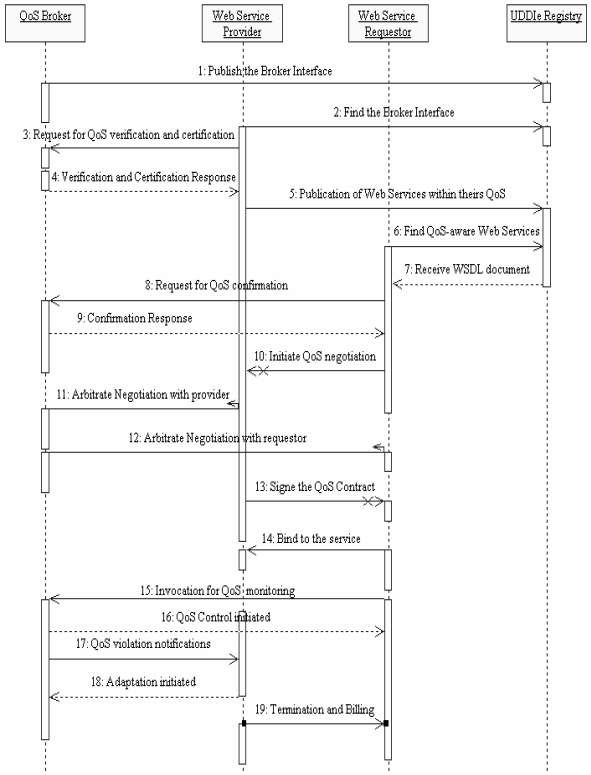

Figure 2. Architecture component interactions

3.1.1. Web services broker. The web services broker assists clients in selecting web services based on a set of QoS parameters. The broker is a web services performing a collection of QoS functionalities. It is the entity that performs the verification and certification tasks. It is also involved in other operations, such as QoS negotiation, monitoring, and adaptation.

3.1.2. Web services provider (server). The provider is the entity that develops the web service and describes its functionalities in addition to the QoS it provides.

3.1.3. Web services client. The client application operates as a service consumer of the advertised web services

3.1.4. UDDI enabled QoS registry. UDDIe is a registry that supports QoS aware web services publication and discovery [15]. It supports the notion of "blue pages", to record user defined properties associated with a service, and to enable discovery of services based on these.

3.2. QoS support in web services

With the integration of Web Services as a business solution in many enterprise applications, the QoS presented by Web Services is becoming the main concern of both service providers and clients. Providers need to specify and guarantee the QoS in their web services to remain competitive and achieve the highest possible revenue from their business. On the other hand, clients aim to have a good service performance (e.g. high availability, short response time, etc.).

3.2.1. QoS parameters for web services. QoS for web services represents the non-functional aspects of the service being provided to the web service users. A wide variety of QoS parameters for web services have been presented in pervious work ([3][4][5][6][7]). For the sake of our experiments, we will consider the following QoS attributes:

Response time (RT): is the time a service takes to respond to the client request. This attribute is measured at the client side and represents the difference between time of sending the request and the time of receiving an answer.

Service charge: is the cost involved in requesting the service. The web service cost can be estimated by operation or by volume of data.

Availability: the probability that the service is accessible (available for use) [3] or the percentage of time that the service is operating [4].

Latency: time taken between the time a service request arrives and the time the corresponding response is generated [7]. This metric is computed at the provider side.

Reputation: is a measure of service trustworthiness. It depends on end user's experiences of using the service. The value of reputation is given by the average ranking given to the service by end users [4]; for example, in Amazon.com, the range is [0,5].

3.2.2. Differentiated class of web service. We defined classes of web services as proposed in [10] to allow a differentiated QoS for different client's profiles. Each class is described by a set of QoS attributes a web service can offer. It exposes different QoS attributes with different values. Table 1 describes an example of QoS classes of a web service according to a set of QoS attributes.

Table 1. Differentiated class of services

Class of web services / QoS Parameters	Class 1	Class 2	Class 3	…	Class n
Response Time	N/A	0.7 ms	0.5 ms		0.1ms
Latency	N/A	N/A	0.1 ms		0.01 ms
Availability	N/A	N/A	0.8		1 (100%)
Reputation	N/A	N/A	N/A		5/5
Service charge	0.10 $	0.2 $	0.25$		0.35$

N/A: not applicable.

4. QoS broker verification and certification model

Verification and certification are keys differentiators of the proposed broker compared to existing approaches ([7][11][12][13]). Web services providers request the QoS broker for QoS certification before publishing their WSDL with QoS classes in UDDIe registry. Before issuing a certificate, the web service should pass a list of verification tasks. In the following subsections, we describe the verification and certification functions and show how they are used to improve the utilization of web services.

4.1. Verification scenarios

The verification process is initiated by the service provider through the "invokeBroker" operation of the web service verifier. During the invocation the web service provider supplies the verifier with its WSDL document and additional information about resources available at the provider platform (operation 1 in figure3). Then, the verifier sends this document to the WSDL parser. We developed a parser application that extracts all useful information from the service interface including the QoS properties (operation 2) and stores them in its database (operation 3). This information consists of a service name, its location, its implementation description, the QoS properties names, types and values. The next operation performed by the service verifier is to test the service URI, the XML schema definition, the service binding information, and the availability of all operations described in the

service interface (operation 4). The service verifier checks also if all the operations described in the service interface are available.

The verifier goes beyond the above verification functions and performs as well the verification of the QoS information introduced in the service interface. QoS verification is conducted through a set of test cases generated by the service verifier to verify the conformity of QoS properties claimed by a provider. To perform each test case the verifier asks for additional information about the provider and its web service (Server capacity, Network bandwidth, etc.). QoS verification process is detailed in the implementation section and includes the verification of Response Time, Availability, and the Price properties.

Once the verification operation is terminated the verifier stores the verification result in its Database (operation 5). It uses the stored information to generate a verification report as shown in Figure 3 (operation6). The service provider have the access to its verification report via a web site and after being authenticated using a specific username and password (operation 7).

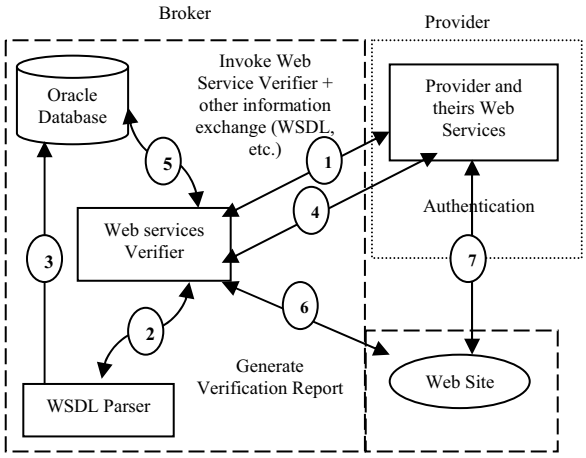

Figure 3. Verification scenarios

The verification process deals with three verification levels: general web services information validation, WSDL document content validation, and QoS description validation. A web service is said to be compliant with a given level when it passes the corresponding set(s) of tests described in the verification document.

Based on this document, web service is classified for example into one of the followings: Silver web services, Bronze web services, and Gold web services. A Bronze web service is, for instance, a service for which most of the verification scenarios failed. A Silver web service is a service for which more than 80% of verification tests succeeded. A web service is qualified as Gold if all the verification tests succeeded.

4.2. QoS certification

Once the verification is passed successfully, the certification process is initiated. The certification process consists of issuing a certificate to the service provider. These certificate states that the offered QoS are conform to their descriptions. The web service *Certifier* is implemented within the broker and is responsible for certifying web services and their provided QoS. A certificate is sent to the web services provider and a copy is stored in the broker's database for future use. A certificate includes information such as certificate number, certificate issue date, number of years in business, services location. If, for some reasons, a certificate cannot be issued, feedbacks are sent to the provider. This may be due to the provider's resource limitations, to his bad reputation, etc

5. Implementation

To show the applicability of our broker-based architecture for QoS enabled web services selection, we developed a prototype. We implemented the web service verifier and certifier, the WSDL parser, and the broker components. For the sake of testing the verification and certification process, we developed a web services called Tri_Stat. The latest provides a set of statistics and math functions (sorting algorithms, statistic functions, etc.) and it also describes and supports the set QoS metrics describes in section 3.2. A java application has been developed to generate clients that consume the Tri_Stat web service. The prototype was developed using: WebLogic platform 8.1 with service pack 2, that include the application server and the development environment (workshop) [16]. Oracle Database version 9i [17]. UDDIe server that support QoS aware web services [15].

5.1. Verification platform

Figure 4 shows the testing platform and interactions between the components: In the following, we briefly describe these components.

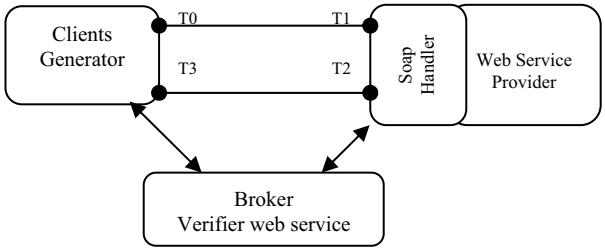

Figure 4. Testing environment

Broker Verifier: is designed as a web service. It tests and verifies the QoS properties of a web service (Response Time, Availability, Cost, etc.).

Client generator: is a multithread Java application implemented to generate many instances of clients that invoke the web service. It also computes the final RT value and forwards it to the web services verifier. It initiates a timer at the instance of sending the request to the service (T0) and captures the time stamp once receiving the answer (T3). The response time value is the difference between these two time stamps (T3 – T0).

Soap Handler: is a Java application developed by the broker and integrated with the provider service to intercept SOAP messages coming from clients. The handler measures the time consumed in processing each client request. It also forwards the processing time (T2-T1) to the web services verifier who uses it to compute the time consumed by the message in transiting the network.

Web Service Provider: is the hosting environment where web services are deployed and available for use by clients.

5.2. Experiments

Our simulation model consists of a single broker, a single web service and N concurrent clients. We measured the RT and the availability attributes measured the below equations (1, 2, and 3).

(1) *RT = T3* – T0 Equation (1) can be rewritten to include the network round-trip and the processing delays as:

(2) *RT = (T1 – T0) + (T2 – T1) + (T3 – T2)*

(3) Availability (s) = <uptime> / <total-time>
= <uptime> / (<upTime> + <downtime>)

The *uptime* is total time the service has been up during the measurement period. The *downtime* is the total time the service has been down during the measurement period. And the *total-time* is the total measurement time.

We propose an approach to generate test cases for three verification scenarios of RT and availability properties. Each scenario takes into consideration resources that may affect the evaluation of the above QoS attributes. These resources might include the network throughput, number of clients connected to the service, the provider and the client server resources capacity (Memory, CPU). Description of each scenario and its related results are illustrated below.

Scenarios 1: We generate a set of concurrent clients and we invoke the broker to calculate the RT and the availability of the service. We increase the number of clients until we reach the server capacity. The objective of this experiment is to check if the RT is stable with the increased load. The network connection and the available resources at the client and the provider are very limited.

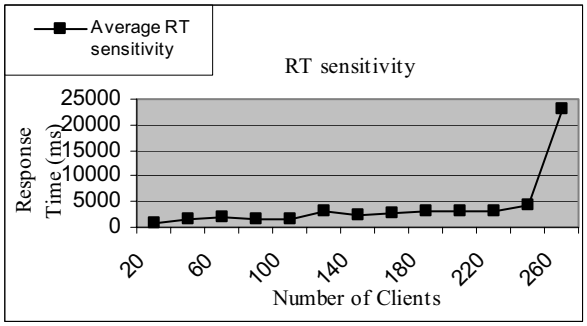

Figure 5. Distribution of RT with increased number of client

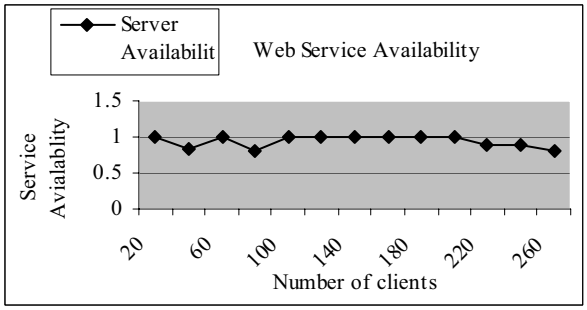

Figure 6. Distribution of availability with increased number of client

Scenarios 2: the client application, the broker and the web service are deployed on different networks locations (LAN, Wireless) and executed at different period of the week. We instantiate the clients, the broker, the web service from different network location and we measure the RT and the availability properties at different period of the week. These experiments are performed during the week end, and in a light load. The objective of this experiment is to check if the RT and availability are preserved with the variation of network resources, the server load and the period of evaluation.

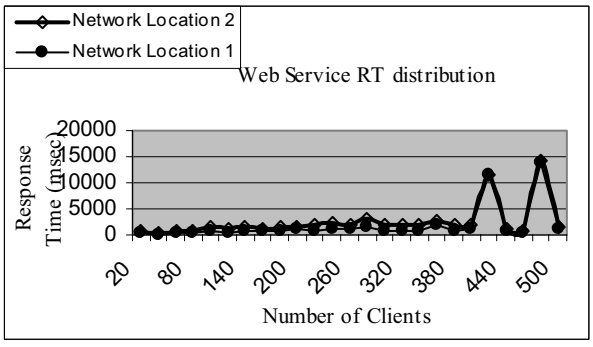

Figure 7. Service RT in a low load conditions

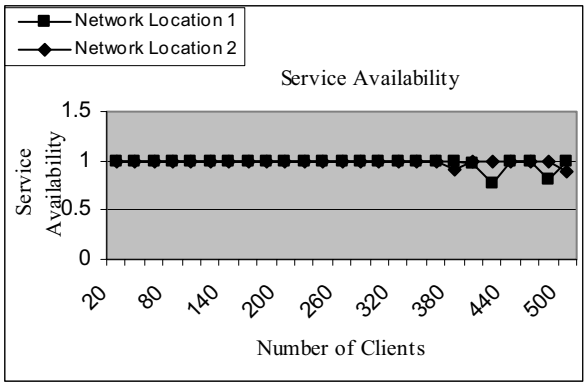

Figure 8. Service availability in a low load conditions

Scenarios 3: we use a limited resource capacity at client, the provider and the broker platform (limited CPU capacity, Dial UP Connection, and limited memory size) and we try to initiate the broker to evaluate the RT and availability under these constraints.

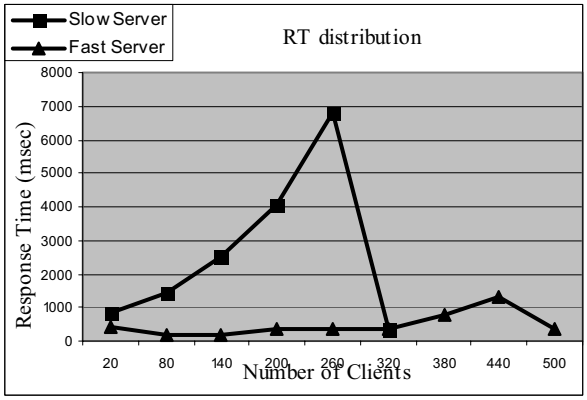

Figure 9. RT distribution for different server capacity

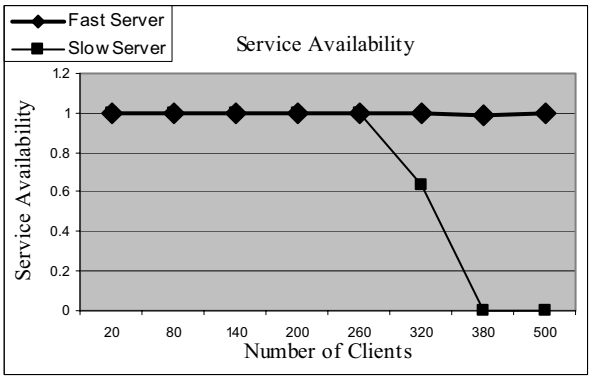

Figure 10. Service availability under different server capacity

5.3. Results and analysis

The result obtained from scenario 1 and shown in figure 5 demonstrates that the response time increase linearly with the number of clients until it saturates at 280 active clients. Figure 6 shows that the service availability is fluctuating with the first triggered clients; and then the service remains soundly available until it becomes 85% available at 220 clients. Latterly the web service becomes unavailable at 280 connected clients.. We conclude from this scenario that the number of client, the network connexion, the service resources capacity have a significant effect on response time and the availability of the service.

Scenario 2 shows that for high speed network connection and higher resource available on the service provider and the client's platform; the service can support until 500 clients from two locations. Under a light load condition and using different network location and significant resources at the client and the provider platform; the service RT is sensibly small and stable with the increased number of clients (figure 7). Alternatively the service availability is very high and stable with the increased number of connected clients (figure 8). Afterwards it decreases sensitively at about 450 clients connected from two network location.

From scenario 3 we conclude that the slowest server has a significantly larger response time and smaller availability than the fastest server (figure 9 and 10). The service reject all receives request and stop responding when it simultaneously deals with more than 320 clients from different locations. Similar behaviours are observed for the service availability that starts to be partially available at 260 active clients from each location and become totally unavailable at 380 clients from each location.

Finally, the results of validation test cases show significant influence of the server resources capacity, the number of connected client, the network load on the RT and the availability of a web service. The result indicates that under light service load, delivery of QoS for clients at different locations has no big difference and all clients are satisfied. When the service is overloaded clients with faster network connection and less network overload have faster and more stable responses. The QoS values computed from the above experiments when compared with the described one are still valid under the applied constraints. However, the broker will exploit the results of these experiments to evaluate the RT and the availability of the service to its provider.

Validation of the other QoS attributes (price, and reputation) described in table 1 is also achieved by the broker. The service verifier store in its database all

important QoS information published in the service interface. Then, it retrieves QoS information of web services that offer the same functionalities. After that, it analyses and evaluates the service charge and reputation according to similar web services offering the same properties. Based on this analysis the service verifier can decide about the conformity of these QoS to the service description. The validation of the latency property is performed using the same architecture described in figure 6 and can be measured using the equation 2 stated in section 5.2.

6. Conclusion and future work

In this paper, we presented a QoS broker-based architecture for web services. The goal of the broker is to support web services QoS verification, certification, confirmation, selection and monitoring. We described the key features of the broker that are not supported by existing approaches dealing with QoS for web services. The main contribution concerns the design of the broker that can be invoked by interested requesters when developed and published as a web service. We emphasize in our work more on the verification and certification process, and we used a methodological approach to measure the QoS attributes and generate test cases for the verification purposes. Also, we illustrated the applicability of the architecture roles with prototype implementation.

We are convinced that the proposed architecture is a good starting point for QoS management of web services. The service provider does not have to design and develop her/his own broker but just invoke one from the published brokers. The client will also find a good support during its web services selection using the broker services.

The main weakness of the architecture is the cost of its adoption. In fact, the broker should be fully operational and its interface has to be known in advance to the providers and clients. However, these limitations are weighted against the benefits in terms of QoS guarantees, and monitoring. We are working in enhancing the proposed architecture to support independent set of broker. These QoS broker will compete collectively in delivering QoS management for providers and clients of web services. This will enable a more flexible, and trustable architecture. Results of this work will be reported in a future paper.

Acknowledgments

The authors would like to thank Abdelmoujoud Lakhlifi, from University of Montreal, for his help developing and running the simulations.

7. References

[1] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[2] D. Gisofli, “Web services architect: An introduction to dynamic e-business”, *IBM paper*, April 2001.

[3] Daniel A. Menascé, “QoS Issues in Web Services”, *IEEE Internet Computing*, December 2002.

[4] Liangzhao, “Quality Driven Web Services Composition”, *The Twelfth International World Wide Web Conference* , Budapest, Hungary, May 2003.

[5] A. Mani and A. Nagarajan, “Understanding quality of service for web services”, *IBM paper,* January 2002.

[6] Ying Li, Xiaochen DING, Ying CHEN, Dong LIU, Thomas LI, “The Framework Supporting QoS-enabled Web Services”, *IEEE International Conference on Web Services*, Las Vegas, Nevada, USA, June 2003.

[7] Shuping Ran, “A Framework for discovering web services with Desired Quality of Services Attributes”, *IEEE International Conference on Web Services*, Las Vegas, Nevada, USA, June 2003.

[8] DAML-S Coalition, DAMLS-S, “Web Service Description for the Semantic Web”, *In Proceeding of the International Semantic Web Conference*, June 2002.

[9] A. Keller and H. Ludwing, “The WSLA framework: Specifying and Monitoring Service Level Agreements for Web Services”, *IBM Research Report*, May 2002.

[10] V. Tosic, B. Pagurek, K. Patel, “WSOL A Language for the Formal Specification of Classes of Service for Web Services”, *International Conference on Web Services*, Las Vegas, Nevada, USA, June 2003.

[11] Hongan Chen, Tao Yu, Kwei-Jay Lin, “QCWS: an implementation of QoS-capable multimedia web services”, *IEEE Fifth International Symposium on Multimedia Software Engineering*, December 2003.

[12] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller, “A Concept for QoS Integration in Web Services”, *4th International Conference on Web Information Systems Engineering*, Rome, Italy, December 2003.

[13] M. Tian, A. Gramm, H. Ritter, J. Schiller, “Efficient selection and monitoring of QoS-aware web services with the WS-QoS framework”, *IEEE/WIC/ACM international Conference on Web Intelligence,* Beijing, China, September 2004.

[14] W.T. Tsai, R. Paul, Z. Cao, L.Yu, A. Saimi, B. Xiao, “Verification of web services using an enhanced UDDI server” *Eighth IEEE International Workshop on Object-Oriented Real Time Dependable Systems*, Guadalajara, Mexico, January 2003.

[15] Ali Shaikhl Ali, Omer F. Rana, Rashid Al-Ali, David W. Walker, “UDDIe: An Extended Registry for Web Services”, *Workshop on Service Oriented Computing: Models, Architectures and Applications*, IEEE Computer Society Press, Florida, USA, January 2003.

[16] BEA WebLogic platform, http://www.bea.com

[17] Oracle Database http://www.oracle.com