Web Services Planning Concepts

Peter FARKAS
Senior student
Budapest University of
Technology and Economics
Department of Automation
and Applied Informatics
H – 1521 Budapest, Hungary
pfarkas@avalon.aut.bme.hu

Dr. Hassan CHARAF, Ph.D.
Associate professor
Budapest University of
Technology and Economics
Department of Automation
and Applied Informatics
H – 1521 Budapest, Hungary
hassan@avalon.aut.bme.hu

ABSTRACT

Building distributed applications by using components from different sources may lead to many problems: such as reliability, confidentiality and quality. The world of web services is dealing with the same disadvantages. To obtain the proper component is solved: use the UDDI database and select the application. But when you need something reliable, the "yellow pages" are not capable of providing information about which component can be built in. Our solution provides additional information about the component quality factors and gives opportunity to query for QoS-enabled application using UDDI database.

Keywords

XML Web Services, QoS, capacity planning

1. INTRODUCTION

With the widespread of web service the only differentiating factor become service usability and utility. Condition of independent messaging is using standards for service components development. Application assemblers, who want to make reliable applications, cannot afford to build in web service components because of its unpredictable availability. From now on the popularity of Web Services only depends on its quality.

2. WEB SERVICE MODEL

The model developed by IBM [WSCA] follows the typical Broker architecture. Web services based upon three roles: service provider, service requestor and service registry. Among them the following interactions can be built: find, publish or bind. The three roles and their interactions determine the Web Services artifact. All of the web services follow the same pattern. One participant can be left out from the model at private web service, the service registry. At this time the service description is published via e-mail, fax, etc.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972
WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

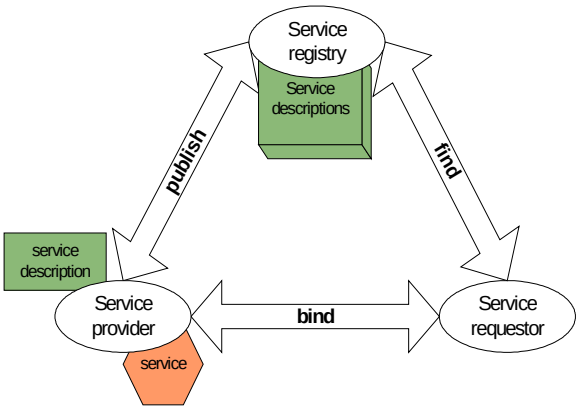

Figure 1 Basic web service roles and interactions

Model's quality expansion

The basic model of web services does not contain any information about quality measurements. To keep accessibility open to any platform, the QoS must be built into this model. All of the roles and interactions are being discussed in the point of quality view, for basic information, please read the reference.

2.1.1 Service provider

This is the server side component of the web service. The owner of the server component is responsible for the service accessibility, availability and its

maintenance. Above all of these it determines quality factors and price of QoS-aware usage.

2.1.2 Service requestor

On the client side we search for certain function that satisfies our needs. But we also need insurance for the quality factors. So not only the service description must be obtained but the quality descriptor as well.

2.1.3 Service registry

In the point of QoS view the registry does not have any effect on the service. It stores the service descriptions and hosts requests for the specified queries.

3. WEB SERVICES' BOTTLENECKS

The performance of the web services is limited by the platform, the operating system and the background services, but the underlying protocols as well. Here are some examples that may affect on the quality of web service [QoS4WS]:

- Web server availability and response time
- Application execution time
- Back-end database performance

HTTP

This protocol is a best-effort, stateless delivery service for data forwarding. Two problems can be discovered:

1. No guarantee of packets being transported to the destination
2. No guarantee of the order of the arriving packets

Bandwidth is clearly a bottleneck as number of users and amount of data increase, because packets are discarded when bandwidth is not available. Traditionally many applications assume infinite bandwidth and zero-latency, but this assumption is not real.

In spite of the fact that the IPv6 defines QoS parameters, web services cannot use this feature, because the programmer does not permit to specify which protocol must be used.

XML

The problem with XML lies in its tags. This markup language allows users to build as long tags as they would like, therefore the size of the message can be various. The compression is 400:1 when we use binary representation of the same data instead of XML. Quick transfer is not possible when the message is large, because fragmentation takes time. The use of XML compression is a way to achieve better performance.

SOAP

SOAP uses XML to deliver its envelope between participants. This performance bottleneck can be eliminated by XML compression. The performance is degraded by:

1. Extracting SOAP envelopes from SOAP messages is time expensive
2. No XML data optimalization available
3. Parsing XML data in SOAP envelope also time-expensive
4. SOAP encoding rules must be included in all messages

Processing the envelope XML parser must be loaded and instantiated. Comparing document-parsing time to checking the wellformedness, validity and conversion, latter costs more time. For improving performance use SAX-based parser… it increases throughput, uses less memory and it is more robust.

WSDL, UDDI

These basic standards of web services do not contain any useful information about QoS requirements. So, quality factors must be placed within their extension.

4. QOS-ENABLED WEB SERVICES

The aspect of e-business indicates integration of applications and web services over the Internet. Guarantee QoS on it is a great challenge because of its heterogenity, and unpredictable nature. The most common problem is that a resource is taken away from the program when it needs it. It leads to performance degradation and has affect on quality factors.

QoS refers to non-functional properties of web services such as performance, reliability, availability and security.

QoS support

The most important factor is to discover the proper QoS language, which is able to describe the web services in the point of quality view. The separate QoS language has to answer the following questions:

- What is the expected latency?
- What is the acceptable round-trip time?

4.1.1 Low level support

Only one solution can be found for QoS for web services, a low level support is presented by Microsoft's GXA specification [GXA]. This packet of recommendation uses the possibility of SOAP messages extensibility for defining extra functions such as message routing, coordination, inspection, transaction-support, and security.

4.1.2 High level support

At the UDDI level there is no solution for quality support. We have no chance to determine which web service is able to provide its functionality with quality assurance. We overwhelmed this disadvantage, and made a separate QoS language that supports UDDI queries with quality factors in it.

Mathematical representation

Assume the following example: we want to build a distributed application with components from different sources in it. User must query UDDI database to get description files, but how he can choose the proper component among them. And what guarantees the reliability of the selected component. The answer is that we do not know. QoS for web service leads to this formula: $R = \sum_{\forall z} \underline{z}^T \underline{\underline{P}} \underline{z}$, where z indicates the quality factor that needs to be observed with its P weight-matrix. R represents the goodness of the web service in quality. To choose the best QoS-enabled web service, we have to select component which goodness is the lowest: $Opt = \min R$.

What kind of quality factor do we observe? Minimum requirements are: response time, network load and cost.

5. Implementation

To provide quality QoS-enabled web services foremost we need two things: the most important is a description language, that defines the QoS parameters provided by the service provider, and a software architecture which can guarantee the necessary factors.

5.1.1 Description language

Web Service QoS Extension Language (WQEL) is responsible for defining the QoS parameters of the service. This document is the basic negotiation certificate for both participants. This language must be able to describe all of the quality measurement numbers that are described in the previous sections. Requirements:

- Abstract, therefore architecture-independent for its portability
- Simple, easy-to-use, well-arranged
- Extensible for containing implementation dependant properties not only the abstract ones
- Based upon web service standards and protocols, like WSDL
- Can be processed by applications and have to be in human readable format

These leads to the XML-based descriptive language, the WQEL. Because the WQEL consist of service properties of the web service, it is suitable to be close connection with WSDL. The best solution to extend the web service description file with quality features is placing an URL into the WSDL's <*documentation*> part. This URL refers to the document, which contains the quality parameters of the web service.

5.1.2 WQEL schema

The schema's structure is similar to the inner content of WSDL document, but here the quality parameters associated with service interfaces. The main goal of this descriptive language that is to correspond to the basic requirements, and to be extensible as well. There are some properties that belong to the service, while others are defining QoS features beneath operations.

Parameter	Belongs to the part of	
	service	operation
Security	√	√
Transactional	×	√
Cost	√	√
Response time	×	√
Availability	√	×

Table 1 Appearance of QoS criteria

Referring to Table 1, we assume that <*service*> part consists of three parts: security (authentication, encryption), cost (component's end-user price), availability (probability that service is available).

Every record in the <*operation*> part corresponds to one operation from the service itself. Four optional parameters are supported at this level: security, transactional (supports transactional mode), cost and response time (minimum, maximum and average).

Parameters can be divided into two: numerical data or not. Numerical factors – like cost, response time, availability, etc. – can be part of the optimalisation algorithm to rank the selection. The others can be used for obtaining minimal information about the web services to filter out unsuitable ones.

We use abstract definitions instead of concrete representation because the language has to be extensible.

5.1.3 QoS architecture for web services

So, if we have the QoS-enabled web service, now we want to search for it. But to this operation we need a new architecture, which query for web services through UDDI databases and discovers the QoS-aware ones. It is much more easier to choose the corresponding resultset which satisfy the requested

quality criteria. Web service QoS Architecture (WQA) requirements are the following:

- Based-upon standards for easy integration
- Collaboration with simple web services and applications

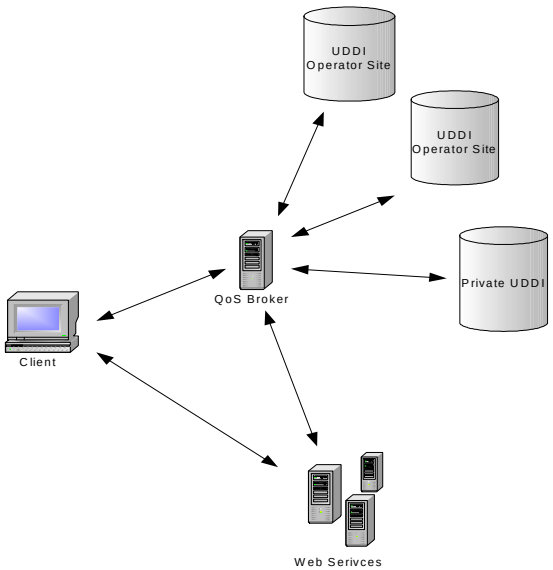

Figure 2 WQA structure

This architecture uses UDDI databases to get WSDL information about web services. But these databases do not support QoS parametric queries, so we need something, that processes those type of requests. This server side component have to be programmable via API from the client side to reach its functionality.

This API extends UDDI Inquiry functions with two methods, which are responsible for QoS specific queries. This component is called *QoS Broker*.

Typical scenario for requesting a QoS-enabled web service:

1. Client application requests the list of QoS Brokers from the UDDI database
2. Selects one and connects to it
3. Sends a QoS-aware query for the broker
4. QoS Broker connects to UDDI database
5. The broker collects all web services which can provide the significant functionality
6. Filters those components that can provide the requested quality factors
7. On the resultset processes an algorithm to choose the optimum services
8. Broker returns the WSDL descriptors to the client

5.1.4 QoS Broker

This parser stands in the center of the architecture. Its role is to choose the best, available web service component from the filtered ones to be built in. Two tasks should be processed: at first it has to discover all of the web services which functionality is fitting to the request. Secondly, on the resultset provided by UDDI query, filters the QoS-aware services. It chooses the optimum web service from the degradated resultset.

But it has to handle standard UDDI request, so the implementation of the UDDI Inquiry functions need to be made.

UDDI Inquiry API	QoS Broker API
find_binding	-
find_business	find_business_qos
find_service	find_service_qos
find_tModel	-
get_bindingDetail	-
get_businessDetail	-
get_businessDetailExt	-
get_serviceDetail	-
get_tModelDetail	-

Table 2 Extended UDDI API

We have to add two new methods, which correspond to the QoS queries. Only two places exist where modifications should be made for quality factor registration in the UDDI model [UDDI]: the business and the service entity.

6. ACKNOWLEDGMENTS

Special thanks to my classmate, Gábor Csorba for testing our solution and giving advises where the schema needs to be modified.

7. REFERENCES

[GXA] Microsoft Global XML Web Services Architecture http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dngxa/html/gloxmlws500.asp

[QoS4WS] Understanding quality of service for Web services http://www-106.ibm.com/developerworks/library/ws-quality.html

[UDDI] UDDI version 2.0 API Specification http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf (PDF format)

[WSCA] Web Services Conceptual Architecture 1.0 http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf (PDF format)