QoS enhanced Web Service selection techniques

Vassiliki Diamadopoulou[2], *Christos Makris*[1,2], *Yannis Panagis*[1,2] *and Evangelos Sakkopoulos*[1,2]

[1] Research Academic Computer Technology Institute
Internet and Multimedia Technologies Research Unit
N. Kazantzaki Str. 26504 Patras, Greece

[2] Department of Computer Engineering & Informatics
School of Engineering, University of Patras
Rio Campus, 26500 Patras, Greece

E-mail: {diamadop, panagis}@ceid.upatras.gr, {sakkopul, makri}@cti.gr

ABSTRACT

This work proposes a Web Service (WS) discovery model in which the functional and non-functional requirements are taken into account during service discovery. The proposed infrastructure includes a set of services and tools to support an integrated WS discovery and selection solution. A mechanism is introduced that supports three different functional policies. It is able to take advantage of quality information located into a WSDL description file that might be located in a proprietary UDDI registry server or in an independent URL. Moreover it implements a database supported WS intermediary (Broker) that it is also possible to store QoS information for Web Services. A selection module is also presented that delivers the WS that maximizes the value of QoS characteristics among others with the same functionality. An experimental prototype is presented and evaluated in the scenario of air ticket issuing WS selection.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval], Information Search and Retrieval - Selection process

H.3.4 [Information Storage and Retrieval], Systems and Software - Information networks

General Terms

Algorithms, Management, Measurement, Documentation, Performance, Design.

Keywords

QoS, Web Services, Modified WSDL.

1. INTRODUCTION

As the Internet and its related technologies grow, and organizations seek to integrate their systems across departmental and organizational boundaries, a services-based approach to building solutions has evolved. However this approach resulted in problems that they had to do with the usability, the functionality and the simplicity of the services provided. Although many efforts have been made in order to solve these problems, the result was the same: there were hundreds of services available over the Internet but they were hardly used.

For a long time now, the creation and the provision of services from businesses over the Internet have been made in a random way which differed from business to business. Thus, despite the large number of services provided over the Internet, in order to use them, one had to know how each service can be invoked. In fact, one had to check even if they use the same communication protocol and generally, he had to adapt his whole system so as to became compatible with that of the supplier of the service. Many times this was very difficult – if not unmanageable – and still more times the businesses designed their systems so as to avoid collaborations with foreigner sources because of complexity and cost.

However the last few years, this seems to change since almost all the businesses that provide services over the Internet are based on a common technology of design, publication and exploitation of their services, that is known as the Web Services (WS) technology [17][19][20][18][13][6]. This technology that has already been established by the World Wide Web consortium [www. w3c.org], solves the problems of usability and functionality of the services. The Web Services technology defines a large number of specifications and rules with which the creation and providing of services over the Internet is made in a simple way. Thus the services took new dimension and henceforth each business can relatively easily create and provide services over the Internet but it can also use existing services in a simpler way.

Although Web Services technology is a promising solution for addressing platform interoperability and compatibility problems faced by system integrators, the adoption rate has been very slow. There are many factors that may contribute to this slow take up such as lack of security, the quality of communication and the automatic way of discovery and incorporation of existing services in a computational system. Web Services technology has yet to address questions such as: "How I will know the Web Service will meet my performance requirements such as 2 ms response time? Will the service be reliable for my mission-critical system's implementation?". Until these questions have been addressed, it is unrealistic to expect that a business would want to search for a Web Service based on the expected functional requirements in a UDDI registry [3][4] and invoke that service without the assurance of knowing the expected quality of service would be met before hand.

This paper is organised as follows. In Section 2, we outline the need of Web Service discovery based on quality characteristics and propose a solution to that problem. Section 3 presents the general architecture of the proposed model and describes the functionality of the the main component of that model, named as MultiService. In Section 4, we introduce three different approaches for WS discovery based on the QoS characteristics. The first two scenarios which are described in sub-section 4.1, use the QoS characteristics retrieved from the suitably modified WSDL description published in the UDDI registry, while the third scenario, described in sub-section 4.2, refers to the case when the regular WSDL is published to the UDDI registry and then, the QoS information is retrieved from the database Quality. The algorithm that modifies the WSDL description in such a way so that it contains QoS information is presented in Section 5. The components of the proposed model are explicitly presented in Section 6, while in Section 7, we explain how this model can be used in case of ticket booking. The paper is concluded in Section 8.

2. MOTIVATION

This work aims at improving the Web Service discovery by proposing Web Service discovery based on the quality aspects [8] (QoS - Quality of Service) of the desired Web Services. We define QoS as a group of non functional characteristics which influence the quality of the service provided by a Web Service. Until today one searches for Web Services via UDDI registry based only on functional characteristics, ignoring the quality requirements. It is foreseeable that there may be more than one Web Services that share the same functional requirements but have different QoS characteristics. It would therefore be useful for each customer who seeks a service that sells air tickets for example, to have the opportunity of being also informed about the quality characteristics of each one of the services that will be discovered and that all will do the same thing: they will sell air tickets. Thus he could choose to use this Web Service that better meets his requirements. For this reason, we propose Web Services discovery based not only on functional characteristics but also on quality requirements. It should be underlined that there is already much research effort in defining QoS for distributed systems in regard to how it can be expressed for a system and how these requirements can be propagated to the resource manager in order to fulfill these QoS requirements. This effort turned out fruitless since it categorizes and defines QoS from different viewpoints with some overlap between them. Has still not been determined the group of QoS characteristics that is important to distributed systems. It is therefore essential to have a framework capturing the QoS provided by the supplier of the service and the QoS required by the customer and combine them when discovering the Web Service maximized match the required QoS.

In this work, we present a integrated approach of Web Services discovery based on QoS characteristics. More precisely, a mechanism is presented that bares such pieces of information into either WSDL [21] description directly or into a database of an intermediary (as a Web Service Broker [5]). Moreover, it retrieves these quality characteristics from the WSDL description as well as from the database and it processes them, in order to find the service with the maximized QoS characteristics, during the discovery of Web Services that share the same functional characteristics. In this way, Web Service discovery based on both quality and functional characteristics is accomplished. We evaluate experimentally the series of our proposed approaches through for the case of ticket issuing.

3. TECHNIQUES ROADMAP

The architecture of the model proposed consists of six components as it appears in Figure 1. The "heart" of our techniques lies in the *MultiService* component. It is a steering infrastructure that performs WS discovery in UDDI, retrieves QoS information and returns the maximized QoS based WS proposal. MultiService consists of four sub systems:

- The sub system SearchUDDI which communicates with the UDDI registry.
- The sub system QoSExtraction which retrieves the QoS information from a WSDL description.
- The sub system CompareQoS which compares the Web Services based on the QoS information that has been retrieved and furthermore, selects the Web Service with the maximized QoS characteristics.
- The sub system which invokes the RetrieveQoS.

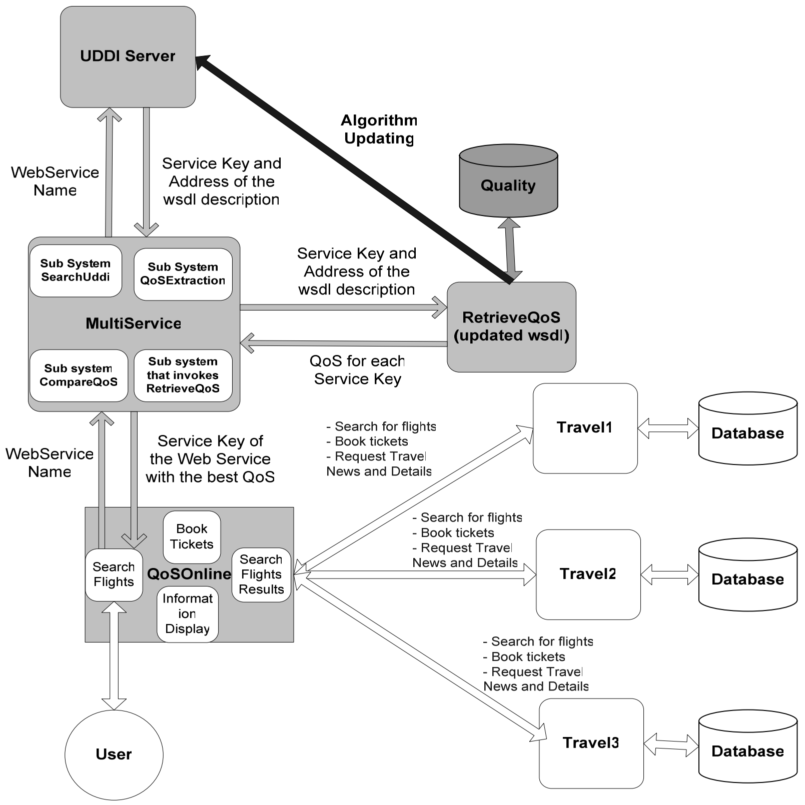


Figure 1. The general architecture of the proposed model

RetrieveQoS is a Web Service that retrieves and delivers the QoS information from various recording and storage sources (in our case a database named Quality).

SearchUDDI communicates with the UDDI registry and retrieves for each service. It then finds, the characteristics of quality for each one of the services discovered:

- either using the sub system QoSExtraction (through WSDL)
- or using the sub system that invokes the RetrieveQoS (through database Quality)

depending on the scenario at hand. All the cases will be explained in detail in section 4. Then, the Service Key of the service with the maximized QoS is returned, which is finally utilized to consume the corresponding Web Service.

4. ARCHITECTURAL OUTLINE

As it was mentioned above, there are two different ways of retrieving the QoS characteristics for the Web Services. Either via the WSDL or via a database. This of course depends on the location where the QoS characteristics are stored. These are either stored to a database which contains for each Service Key the values of the QoS characteristics that belong to it, or they are encapsulated in the WSDL of the service. This file can be modified suitably so that it provides QoS information for a service. The name and the value of such an information are stored as the name of a separate operation, therefore the modified WSDL is the regular WSDL with more operations that correspond to the QoS information of the service. The different approaches for WS discovery based on the QoS characteristics introduce a set of three different techniques that are implemented as separate operation modes of the MultiService intelligent component. Details concerning each functional case follow:

4.1 QoS characteristics via the WSDL

In this case, the modified WSDL has been published in the UDDI registry. We distinguish in more detail two sub-cases:

1. either the QoS values that are found in WSDL are kept,
2. or these QoS values have been stored in UDDI registry a long time ago according to a logged timestamp and they don't represent any more the current QoS that the service has, therefore accessing the database Quality the current QoS can be retrieved.

4.1.1 QoS Extraction Directly from WSDL

In Figure 2, the MultiService accesses the modified WSDL using its URI that was returned and invokes the sub system QoSExtraction. This procedure is repeated for all WSDL addresses that have been returned. In the end the QoS for each service is known and by invoking the sub system CompareQoS, the QoS is compared so that the Service Key of the service with the maximized QoS is selected.

4.1.2 Updated QoS retrieved from Quality DB

In this case, (Figure 3) MultiService retrieves the QoS for each service by invoking the RetrieveQoS which in turn accesses the database Quality where the current QoS that belong to each Service Key is stored. This is repeated for each Service Key that was returned from the UDDI registry and finally, the MultiService is informed about the current QoS for each service and by invoking the sub system CompareQoS, it compares them, selecting the Service Key of the service with the maximized QoS.

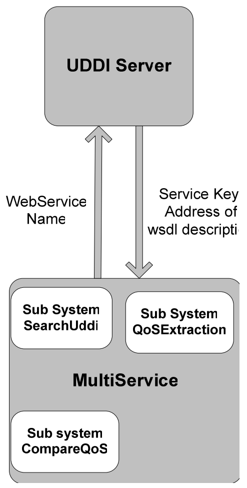


Figure 2. Publish the modified WSDL and QoS discovery from it

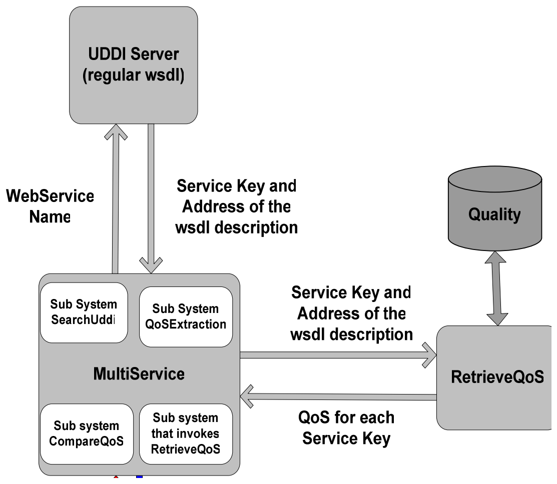

UDDI Server (regular wsdl)
WebService Name
Service Key and Address of the wsdl description
Quality
Sub System SearchUddi
Sub System QoSExtraction
MultiService
Sub system CompareQoS
Sub system that invokes RetrieveQoS
Service Key and Address of the wsdl description
RetrieveQoS
QoS for each Service Key

Figure 4. Publish the normal WSDL description and QoS discovery through database Quality

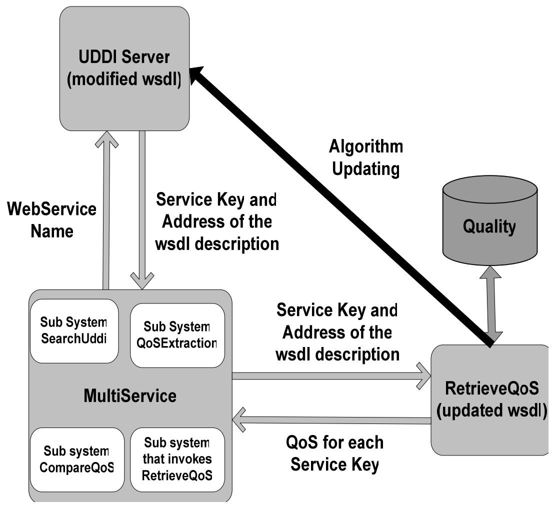

UDDI Server (modified wsdl)
Algorithm Updating
WebService Name
Service Key and Address of the wsdl description
Quality
Sub System SearchUddi
Sub System QoSExtraction
MultiService
Sub system CompareQoS
Sub system that invokes RetrieveQoS
Service Key and Address of the wsdl description
RetrieveQoS (updated wsdl)
QoS for each Service Key

Figure 3. Published the modified WSDL but current QoS from database Quality

There is also the possibility of updating the WSDL description with the current QoS from the database by invoking the RetrieveQoS and of publishing the updated WSDL to the UDDI registry by using the algorithm Updating.

4.2 QoS discovery via Quality DB

In this case, (Figure 4) a typical WSDL has been published to the UDDI registry. The sub system SearchUDDI is invoked, which retrieves for each service its Service Key. Then, the MultiService searches for QoS for each service found by invoking the RetrieveQoS which in turn accesses the database Quality that stores for each Service Key the current QoS values that belong to it. This is repeated for each Service Key that was returned from the UDDI registry and finally, the MultiService is informed about the current QoS for each service and by invoking the sub system CompareQoS, it compares them, selecting the Service Key of the service with the maximized QoS.

5. MODIFYING WSDL

The WSDL description can be modified suitably so that it provides QoS information for the service inherently. The name and the value of such an information is stored as the name of an operation and therefore the modified WSDL is the regular WSDL with more operations that correspond to the QoS information of the service.The algorithm that modifies the WSDL description in the way mentioned above is presented schematically in Figure 5. The algorithm that modifies WSDL.

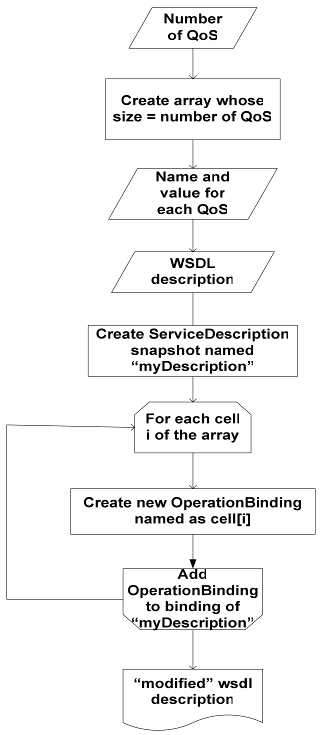
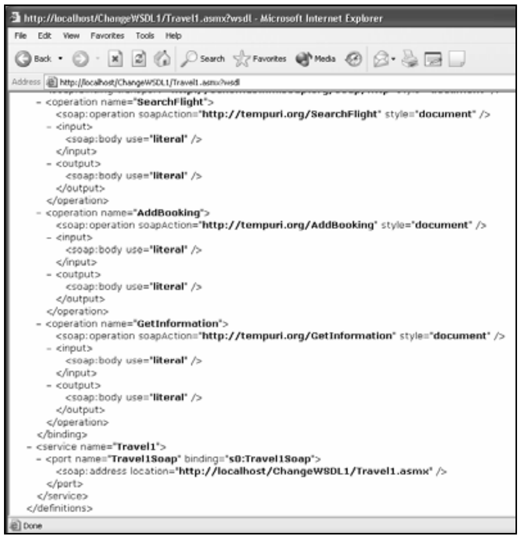

Number of QoS
Create array whose size = number of QoS
Name and value for each QoS
WSDL description
Create ServiceDescription snapshot named "myDescription"
For each cell i of the array
Create new OperationBinding named as cell[i]
Add OperationBinding to binding of "myDescription"
"modified" wsdl description

Figure 5. The algorithm that modifies WSDL

The source code follows:

```
// Obtain the ServiceDescription of existing WSDL.
ServiceDescription myDescription =
ServiceDescription.Read(filename);

Console.WriteLine("Please give the number of the
QoS characteristics :");
int count = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Please give the QoS
characteristics as QoSName_QoSValue :");

string[] qos = new string[count];

foreach(System.Web.Services.Description.Binding b
in myDescription.Bindings)
{
        for(int i=0;i<count;i++)
        {
                qos[i] = Console.ReadLine();

 // Create OperationBinding instance for operation.
            OperationBinding myOperationBinding =
new OperationBinding();
            myOperationBinding.Name = qos[i];
// Create InputBinding for operation.
InputBinding myInputBinding = new InputBinding();
SoapBodyBinding mySoapBodyBinding = new
SoapBodyBinding();

        mySoapBodyBinding.Use =
SoapBindingUse.Literal;

        myInputBinding.Extensions.Add(mySoapBodyBind
ing);

        // Create OutputBinding for operation.
        OutputBinding myOutputBinding = new
OutputBinding();
        myOutputBinding.Extensions.Add(mySoapBodyBin
ding);

// Add 'InputBinding' and 'OutputBinding' to
// 'OperationBinding'.
        myOperationBinding.Input = myInputBinding;
        myOperationBinding.Output = myOutputBinding;

// Create extensibility element for
// 'SoapOperationBinding'.
        SoapOperationBinding mySoapOperationBinding
= new SoapOperationBinding();

        mySoapOperationBinding.Style =
SoapBindingStyle.Document;

        mySoapOperationBinding.SoapAction =
"http://tempuri.org/webservices/";

// Add extensibility element 'SoapOperationBinding'
// to 'OperationBinding'.
        myOperationBinding.Extensions.Add(mySoapOper
ationBinding);

        b.Operations.Add(myOperationBinding);
        }
}
myDescription.Write(filename2);
```

The algorithm is implemented as a .NET application [11] that takes as input the WSDL description that will be modified. The user is asked to give the number of QoS he wants to store to the WSDL. Then, he is asked to give the names and the values of the QoS.



http://localhost/ChangeWSDL1/Travel1.asmx?wsdl - Microsoft Internet Explorer

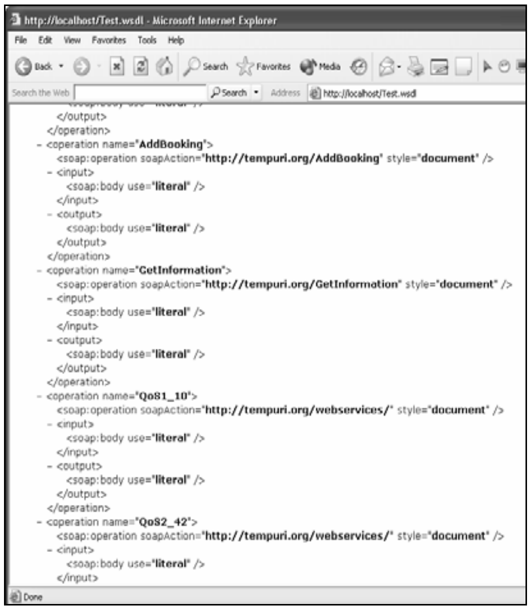
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ChangeWSDL1/Travel1.asmx?wsdl

```
  - <operation name="SearchFlight">
      <soap:operation soapAction="http://tempuri.org/SearchFlight" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="AddBooking">
      <soap:operation soapAction="http://tempuri.org/AddBooking" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="GetInformation">
      <soap:operation soapAction="http://tempuri.org/GetInformation" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
- <service name="Travel1">
  - <port name="Travel1Soap" binding="s0:Travel1Soap">
      <soap:address location="http://localhost/ChangeWSDL1/Travel1.asmx" />
    </port>
  </service>
</definitions>
```

Figure 6. The WSDL description before being modified

The operations in the WSDL description of the Web Service, before being modified by the algorithm that is presented, appear in Figure 6.

Applying the algorithm that modifies the WSDL description in this WSDL description, giving as input two QoS named as QoS1 and QoS2 and their values are 10 and 42 respecectively, it results in a WSDL with two new operations QoS1_10 and QoS2_42, shown in Figure 7. The modified WSDL description.



http://localhost/Test.wsdl - Microsoft Internet Explorer

File Edit View Favorites Tools Help
Back Search Favorites Media
Search the Web Search Address http://localhost/Test.wsdl

```
      </output>
    </operation>
  - <operation name="AddBooking">
      <soap:operation soapAction="http://tempuri.org/AddBooking" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="GetInformation">
      <soap:operation soapAction="http://tempuri.org/GetInformation" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="QoS1_10">
      <soap:operation soapAction="http://tempuri.org/webservices/" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  - <operation name="QoS2_42">
      <soap:operation soapAction="http://tempuri.org/webservices/" style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
```

Figure 7. The modified WSDL description

6. COMPONENTS SPECIFICATIONS

6.1 The service "RetrieveQoS"

This Web Service executes two operations. Accesses database Quality by invoking the function FindQoS that takes as input a ServiceKey and based on this ServiceKey, the QoS values that belong to it are returned. Then, these QoS values are stored to an array but also based on these, the modified WSDL is updated (by invoking the function ChangingWSDL) so that it reflects the current QoS values. This algorithm is presented schematically in Figure 8. The Web Service “RetrieveQoS”. Then, the upadated WSDL is used by the algorithm Updating, that will be presented in the next unit so as to the UDDI registry points to the updated WSDL, when accessed.

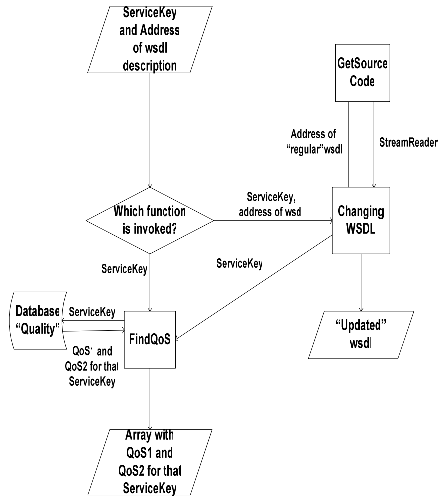

ServiceKey and Address of wsdl description
GetSource Code
Address of “regular”wsdl
StreamReader
Which function is invoked?
ServiceKey, address of wsdl
Changing WSDL
ServiceKey
ServiceKey
Database “Quality”
ServiceKey
QoS' and QoS2 for that ServiceKey
FindQoS
“Updated” wsdl
Array with QoS1 and QoS2 for that ServiceKey

Figure 8. The Web Service “RetrieveQoS”

6.2 The algorithm “UPDATING”

In case when we discover QoS from the database Quality (it contains always the current QoS), there is a possibility of updating the WSDL description that has been published to the UDDI registry with the current QoS. The function ChangingWSDL does that. More specifically, this function creates a file that contains the WSDL description which is now updated with the current QoS from the database Quality and stores it in a suitable place. The algorithm Updating uses that file, and informs the UDDI in such a way, in order that it points to the updated WSDL. In other words, this algorithm updates the OverviewUrl section of the UDDI so that it points to the location where the updated WSDL has been stored. The algorithm is presented schematically in Figure 9. The algorithm Updating.

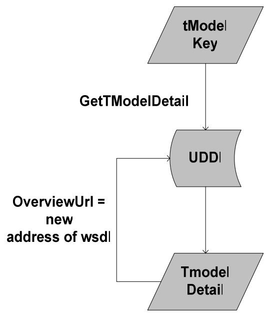


Figure 9. The algorithm Updating

6.3 MULTISERVICE

This Web Service constists of many sub systems: a sub system that implements communication with the UDDI registry, a sub system that invokes the RetrieveQoS, a sub system that retrieves QoS from the modified WSDL and a sub system that compares QoS. Three different possible policies are distinguished which have already been described. Following the various sub systems are discussed in more detail.

6.3.1 SearchUDDI (communication with UDDI)

It implements communication with the UDDI registry and by giving the name of the Web Service. It is more or less a standard UDDI search wrapper.

6.3.2 The sub system that invokes the RetrieveQoS

We distinguish two different cases:

1. either the QoS information is returned and the updated WSDL description is created in case when the modified WSDL is published and is updated through the database quality,
2. or only the QoS information is returned in case when the regular WSDL is published.

The algorithms are depicted schematically in Figure 10 and Figure 11 respectively.

6.3.3 QoSExtraction (QoS discovery from WSDL)

Here, the function GetSourceCode is invoked for each service with parameter the address of the modified WSDL, and then, copies from this address the modified WSDL of the service to a StreamReader who is returned. Then, the StreamReader is read and the QoS found is stored to the array quality. In the end the array quality contains all the QoS that is found in the WSDL descriptions of the services. The algorithm of sub system QoSExtraction is presented schematically Figure 12. The sub system QoSExtraction.

6.3.4 CompareQoS (comparison of QoS)

Upon completion of any of the above cases the QoS information is stored for each WS in the selection candidate list. QoS information values are compared based on their normalized “resulting” value that is computed according to the following concept:

$$\texttt{result} = (\texttt{QoS1}^2 + \texttt{QoS2}^2)^{1/2}$$

The WS that maximizes its normalized “result” value is finally selected. The logic behind the current sub system is outlined at Figure 14.

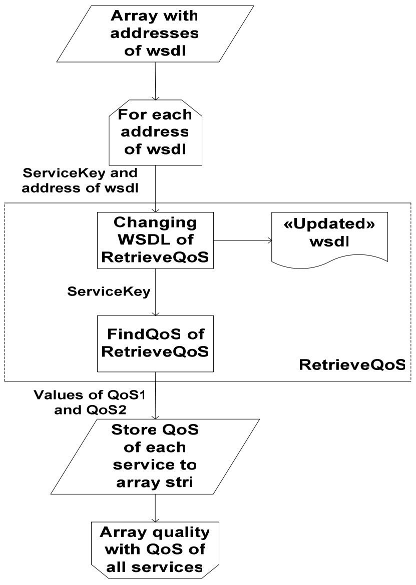

Array with addresses of wsdl
For each address of wsdl
ServiceKey and address of wsdl
Changing WSDL of RetrieveQoS
«Updated» wsdl
ServiceKey
FindQoS of RetrieveQoS
RetrieveQoS
Values of QoS1 and QoS2
Store QoS of each service to array stri
Array quality with QoS of all services

Figure 10. The sub system that invokes RetrieveQoS A΄

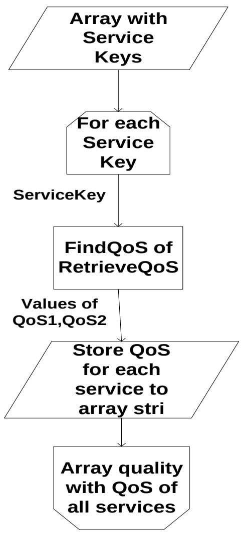


Figure 11. The sub system that invokes RetrieveQoS B΄

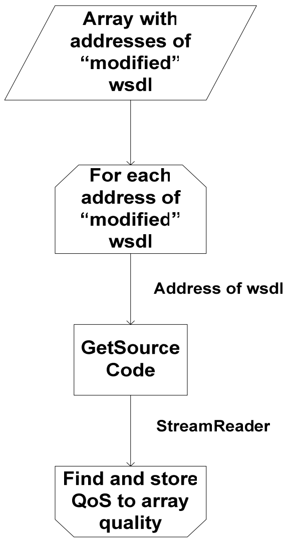

Array with addresses of "modified" wsdl
For each address of "modified" wsdl
Address of wsdl
GetSource Code
StreamReader
Find and store QoS to array quality

Figure 12. The sub system QoSExtraction

7. Experimental Prototype: QoSOnline site

We have implemented our techniques using the .NET v.1.1 framework technology and C# language [1] in particular. Generality is not lost as other Web Service supporting development platform may be utilized. The prototype is hosted on a MS Windows 2003 Server Pentium IV at 3GHz using 512MB RAM. Quality database is designed and runs on an MS SQL Server 2000 sp4 instance. The particular development platform has been chosen mainly because of the authors' previous experience in the area. For the UDDI registry connection the MS online UDDI catalogue available for testing purposes has been utilized at [16].

The QoSOnline involves the following components:

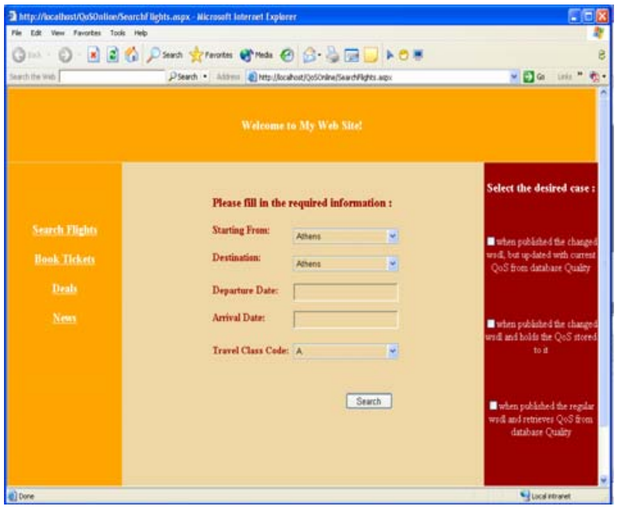
- Demo Web Services Travel1, Travel2, Travel3 that implement air ticket issuing services.
- Utilization of the MultiService infrastructure.
- The Web Service RetrieveQoS
- The intermediary database Quality
- UDDI registry connection.

An evaluating user, who enters the QoSOnline, is asked to complete a search form for air ticket information. Based on the submitted information, the Web Service discovery based on the QoS is fired in order to find flights from the demo available data that satisfy the user's criteria. Then, the flights found are presented to him and then, he can book tickets online. Still, he can be informed on the latest news and deals as far as this service is concerned. The QoSOnline web application consists of four parts:

- SearchFlights – here the user completes a form and selects the way of the QoS discovery
- SearchFlightsResults – it displays the flights found and gives the opportunity of booking a ticket
- BookTickets – it displays the ticket reservation receipt
- DisplayInformation – it displays the latest booking information.

This page allows users to search for flights that satisfy concrete criteria such as the destination place and the departure date, and also gives them the opportunity of selecting the way of retrieving the QoS. The outline of SearchFlights business logic is presented schematically in Figure 16.

The Search button click event handler invokes MultiService infrastructure and provides as parameter the description of the service to be discovered in the UDDI registry. In the experimental prototype the evaluating user may as well choose one of the three different policies – operation modes- available by the MultiService QoS based retrieval component. The way the QoS is retrieved depends on the value of the variable "periptwsh" which takes a different value depending on the case selected. A case can be selected by clicking one of the three CheckBox available on the page. Then, the Service Key of the appropriate service with maximized QoS information is stored as well as the QoS of all the services found for statistical purposes. Finally, SearchFlightsResults displays the flights found.


http://localhost/QoSOnline/SearchFlights.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites Media
http://localhost/QoSOnline/SearchFlights.aspx
Welcome to My Web Site!
Search Flights
Book Tickets
Deals
News
Search
Done
Local intranet

Please fill in the required information :

Starting From: Athens
Destination: Athens
Departure Date:
Arrival Date:
Travel Class Code: A

Select the desired case :

☐ when published the changed wsdl, but updated with current QoS from database Quality

☐ when published the changed wsdl and holds the QoS stored to it

☐ when published the regular wsdl and retrieves QoS from database Quality

Figure 13. The SearchFlights interface

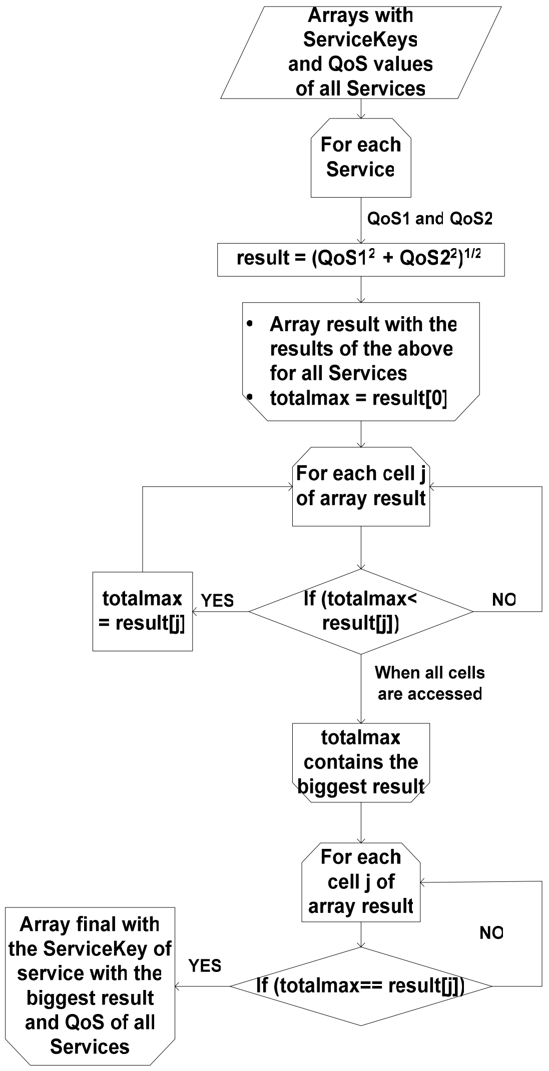
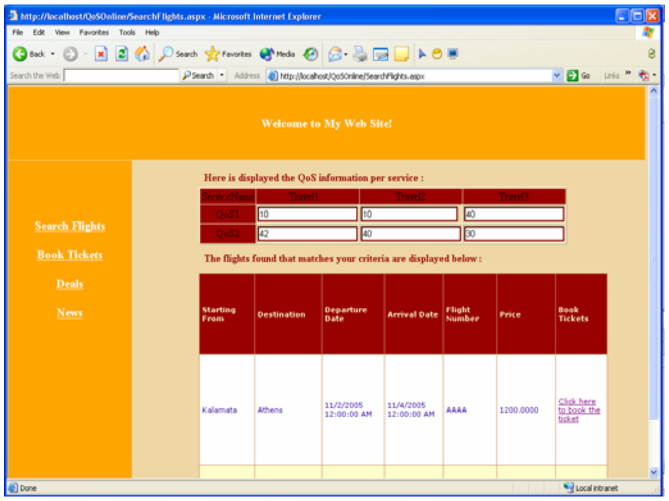

Arrays with ServiceKeys and QoS values of all Services
For each Service
QoS1 and QoS2
result = $(QoS1^2 + QoS2^2)^{1/2}$
• Array result with the results of the above for all Services
• totalmax = result[0]
For each cell j of array result
totalmax = result[j]
YES
If (totalmax< result[j])
NO
When all cells are accessed
totalmax contains the biggest result
For each cell j of array result
NO
Array final with the ServiceKey of service with the biggest result and QoS of all Services
YES
If (totalmax== result[j])

Figure 14. The sub system CompareQoS


http://localhost/QoSOnline/SearchFlights.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/QoSOnline/SearchFlights.aspx Go Links
Welcome to My Web Site!
Search Flights
Book Tickets
Deals
News
Done
Local intranet

Here is displayed the QoS information per service :

ServiceName	Travel1	Travel2	Travel3
QoS1	10	10	40
QoS2	42	40	30

The flights found that matches your criteria are displayed below :

Starting From	Destination	Departure Date	Arrival Date	Flight Number	Price	Book Tickets
Kalamata	Athens	11/2/2005 12:00:00 AM	11/4/2005 12:00:00 AM	AAAA	1200.0000	Click here to book the ticket

Figure 15. The SearchFlightsResults interface

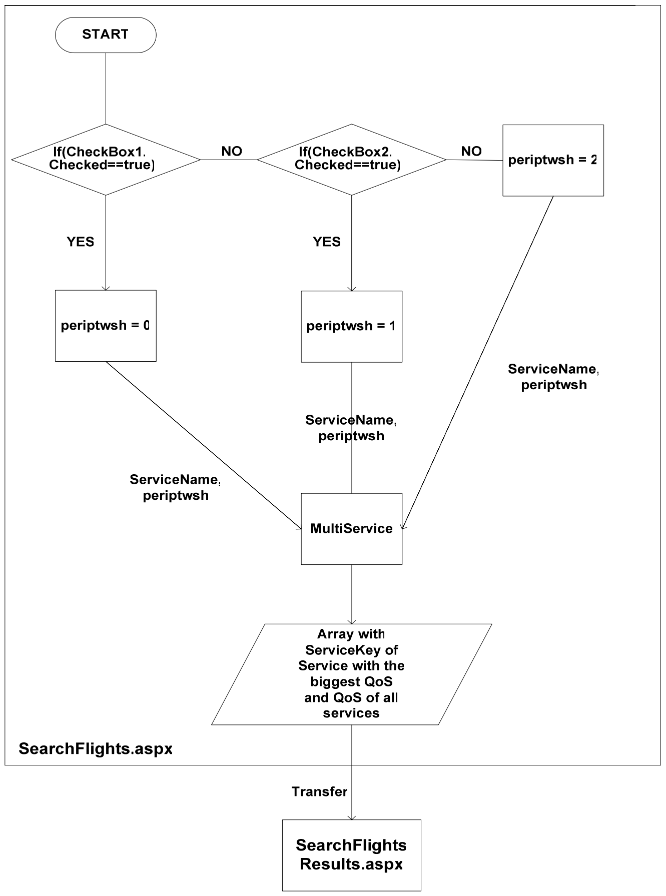


Figure 16. The business logic behind SearchFlights

This page displays the flights found in the service with the maximized QoS and which satisfy the user's criteria placed earlier. The business logic followed by SearchFlightsResults is presented schematically in Figure 17.

Our experimental prototype has been developed to validate and exhibit the potentials of our proposed approaches for a real life test case. Initial laboratory based assessment procedures have verified that the system is effective.

8. CONCLUSIONS AND FUTURE STEPS

The quality-based selection of Web Services is an active research topic in the dynamic composition of services area [9]in general. This work proposes a Web Service discovery model in which the functional and non-functional requirements are taken into account during service discovery. A mechanism is presented that bares quality information into either the WSDL description or in a database of an WS selection intermediary (Broker like). Moreover, it retrieves these quality characteristics from the WSDL description as well as from the database and processes them, in order to find the service with the maximized QoS characteristics among Web Services that share the same functional characteristics. In this way, Web Service discovery based on both quality and functional characteristics is accomplished. QoS based enhancement in the WS selection process elevates the effectiveness of the delivered services as certain constraints are fulfilled. Our solution has been experimentally evaluated in laboratory environment and we received encouraging results. The advantages of the proposed model follow in the list below.

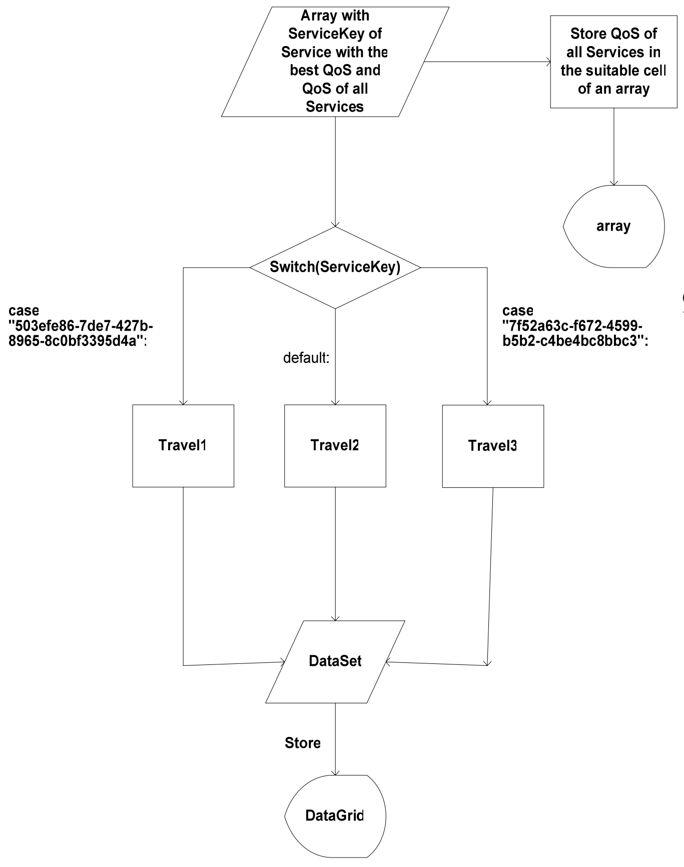

Array with ServiceKey of Service with the best QoS and QoS of all Services
Store QoS of all Services in the suitable cell of an array
array
Switch(ServiceKey)
case "503efe86-7de7-427b-8965-8c0bf3395d4a":
default:
case "7f52a63c-f672-4599-b5b2-c4be4bc8bbc3":
Travel1
Travel2
Travel3
DataSet
Store
DataGrid

Figure 17. The business logic behind SearchFlightsResults

- The model presented supports all three different QoS retrieval scenarios: QoS retrieval from a database that we call *Quality DB* in cases that a typical -"normal"- WSDL is already published, latest - updated QoS information retrieval from database when a "modified" WSDL already containing QoS information is published (WSDL QoS info is considered expired), QoS retrieval from the "modified" WSDL itself when it is published.
- The model presented ensures that the QoS of published Web Services remain open by retrieving the current QoS from the database Quality each time a service is invoked in the two cases when the database is used.
- A mechanism that informs service providers about the QoS computed for their services and updates their published services so as to become more competitive at any time.
- The end users are informed about the QoS information of all the services they asked for.

Future steps indicating next research attempts and directions in the area can be found in the following.

- A mechanism that assigns trustworthy and fair QoS to services and controls them periodically so as to update the values of the QoS first assigned to them. This mechanism stores the QoS values to database Quality and updates it so as it always contains the current QoS.
- A mechanism which ensures that the QoS of the published Web Services remain open, trustworthy and fair in case when we retrieve QoS from the "modified" WSDL description (not using database Quality).
- A sufficient group of QoS that can describe a Web Service and a set of metrics to quantify each one QoS.
- Utilization of semantically annotated WSDLs for more complex context sensitive QoS constraints fulfilment.

9. REFERENCES

[1] Ashish Banerjee, Aravind Corera, Zach GreenVos, Andrew Krowczyk, Brad Maiani, Christian Nagel, Chris Peiris, Thiru Thangarathinam, *C# Web Services: Building Web Services with .NET remoting and ASP.NET,* Wrox Press, 2001.

[2] Shuping Ran, *A Model for Web Services Discovery With QoS,* CSIRO Mathematical and Information Sciences, ACM SIGecom Exchanges, 2003.

[3] UDDI COMMITTEE, UDDI Version 2.03 Data Structure Reference, http://www.UDDI.org/pubs/DataStructure v2.htm, 2002.

[4] OASIS, Universal Description Discovery and Integration of Web Services (UDDI) Version 2, http://www.UDDI.org, 2002.

[5] Tao Yu, Kwei-Jay Lin, *The Design of QoS Broker Algorithms for QoS-Capable Web Services,* Dept. of EECS, University of California, IEEE 2004.

[6] Peter Farkas, Hassan Charaf, *Web Services planning concepts*, Journal of WSCG, 11(1), February 2003.

[7] C. Tsetsekas, S. Maniatis, F. Fünfstück, A.Thoma, Y.Karadimas, *A QoS middleware between users, applications and the network*, ComCon 8 (8th Intl. Conf. on Advances in Communications and Control), Crete, Greece, June 25-29, 2001.

[8] Daniel A. Menasce, *QoS Issues in Web Services,* IEEE Internet Computing, vol.6, no.6, pp. 72-75, Nov./Dec. 2002.

[9] J.P.Hansen, J.Lehoczky and R.Rajkumar,*Optimization of Quality of Service in Dynamic Systems,* in Proc. of the 9th International Workshop on Parallel and Distributed Real-Time Systems, April 2001.

[10] Yutu Liu, Anne H.H. Ngu, Liangzhao Zeng, *QoS Computation and Policing in Dynamic Web Service Selection*, WWW 2004.

[11] Tom Archer, Andrew Whitechapel, *Inside C#,* second edition, Microsoft Corporation.

[12] Microsoft, *Application Architecture for .NET: Designing Applications and Services*.

[13] Keith Ballinger, *.NET Web Services: Architecture and Implementation*, Addison-Wesley 2003.

[14] Intoduction to Microsoft SQL Server and SQL Code Examples, www.functionx.com/sqlserver/index.html.

[15] Online C# lessons, www.functionx.com/csharp/index.htm

[16] Microsoft UDDI Business Registry Node – Test Site http://test.UDDI.microsoft.com/

[17] Kamalsinh, Kennesaw, Kennesaw, *Anatomy of a Web Service,* CCSC: Southeastern Conference, 2003.

[18] Ewald, Tim. (September 27, 2002), *Understanding XML Web Services: The Web Services Idea,* Retrieved March 20, 2003 from http://msdn.microsoft.com/webservices/understanding/readme/default.aspx.

[19] *Introduction to Web Services Part I.* (n.d.). Retrieved February 23, 2003 from http://www.epionet.com/webservices/articles/art_webservicesintro1.html.

[20] *Introduction to Web Services Part II*. (n.d). Retrieved February 23, 2003 from http://www.epionet.com/webservices/articles/art_webservicesintro2.html.

[21] *WSDL Tutorial*.(n.d). Retrieved April 9, 2003 from http://www.w3schools.com/WSDL/default.asp.