Service Selection Using Quality Matchmaking

Amna Eleyan , Liping Zhao
Birzeit University, University of Manchester
Palestine, United Kingdom

Abstract— **This paper proposes a quality matchmaker which introduces four algorithms or filters: interface matching, quality criteria matchmaking, quality value constraints matching, and mathematical matchmaking. These four algorithms use the quality matchmaker sub-components to implement their roles. The quality matchmaker has three sub-components which are: interface matchmaking, quality criteria matchmaking and mathematical matchmaking.**

A quality matchmaking process (QMP) is introduced to demonstrate the above four algorithms and to select the best Web service. The mathematical matchmaking algorithm is the most important step that uses a mathematical model in order to select the best candidates Web service based on requester's quality requirements and preferences. Two techniques are used in a mathematical model: Analytical Hierarchy Process (AHP) and Euclidean distance.

***Index Terms*—Web services, quality matchmaker, quality matchmaking process, mathematical model**

I. INTRODUCTION

A. Motivation

The Web services technology enables software applications to communicate with each other in a platform- and programming language- independent manner. The Web services technology achieves system interoperability by exchanging an application development and service interactions using the XML–based standards, such as Simple Object Access Protocol (SOAP) [1], Web Service Description Language (WSDL) [2] and Universal Description, Discovery and Integration (UDDI) [3].

As the popularity of the Web services technology grows, the service requester is becoming increasingly aware of the importance of the service quality. Therefore, it is necessary for him/her to have a way of evaluating and selecting the services that meet his/her quality requirement. However, the current Web service technology is immature and still under development by the World Wide Web Consortium (W3C) [4]. and has the following challenges:

1. The service selection in the current Web service architecture is done by human clients, which is not desirable if thousands of services are available for selection.

2. The current selection is only based on the functional information in the WSDL document. The service requester requires a selection mechanism that is based on functional and also the non-functional information. Therefore, an effective automated technique for the service selection regarding to the service requester's quality requirement and preferences is needed.

This paper proposes a quality matchmaker which is the core component of the quality-based Web service architecture (QWSA) [5]. This implements the quality matchmaking process (QMP) to select the best service. The QMP is based on a mathematical model. A simulation programme called the quality service selection system (QSSS) [5] is developed to implement the QMP. It allows the service requester to select the best service automatically.

B. Related Work and Our Contribution

Several research efforts have been made in the area of quality-based Web Services. Zeng et al. [6] present two service selection approaches: local optimization and global planning. A Simple Additive Weighing technique is used to select an optimal Web service. The users express their preferences regarding QoS by providing values for the weights. They propose a simple QoS model using the examples of price, availability, reliability and reputation.

Liu et al.[7] present an open, fair and dynamic QoS computation model for Web services selection. They achieve the dynamic and fair computation of QoS values of Web services through a secure user's feedback and a monitor. Their QoS model is extensible, new domain specific criteria can be added, without changing the underlying computation model. They provide an implementation of a QoS registry based on their extensible QoS model.

Fedosseev in [8] presents the *global planning approach* which is used to optimally select component services during execution of a composite service. This approach is based on quality-of-service (QoS) characteristics of services, different types of quality metrics have been introduced such as QoS: system, QoS: task, quality-of-experience (QoE), and quality-of-business (QoBiz).

This paper proposes a quality matchmaking selection technique that is based upon a mathematical model. The Analytical Hierarchy Process (AHP) is used to calculate the quality criteria weight, based on the requester preferences. The Euclidean distance is used to calculate the distance between the quality requirements and the quality specifications. The service associated with the minimum distance is the best service to select.

II. MATHEMATICAL MODEL FOR SERVICE SELECTION

The quality service selection in this paper depends on the quality matchmaking process (QMP), which is described in Section IV. QMP is based upon a mathematical model. The proposed mathematical model uses two methods in order to select the best Web service. Analytical Hierarchy Process (AHP) method is used to calculate the quality criteria weights based on the service requester's quality preferences. Euclidean distance method is used as in [9], to measure the distance between the quality requirements specified by the service requester and the quality specifications specified by the service provider. The Web service with the minimum Euclidean distance is the best service to select. The mathematical model is described in the following steps using an example.

Step-1: Construct pair-wise comparison matrix

The pair-wise comparison matrix A, equation (1), is constructed with respect to the service requester's quality preferences and compares them in a pair wise way. The pair-wise comparison matrix A is a reciprocal matrix representing the service requester judgments of selecting the relative importance of his preference of quality criterion C_i over C_j from Table 1.

The main diagonal of the matrix is always 1. The requester specifies m(m-1)/2 preferences, where m is the number of quality criteria.

$$\begin{bmatrix} 1 & 12 & 1 \\ 21 & 1 & 2 \\ & & \\ 1 & 2 & 1 \end{bmatrix} \quad (1)$$

Table 1 Relative Importance Measurement Scale [10]

Relative Importance Measurement Scale	
Importance Intensity	**Definition**
9	Extremely Preferred
8	Very strongly to extremely
7	Very strongly preferred
6	Strongly to very strongly
5	Strongly preferred
4	Moderately to strongly
3	Moderately preferred
2	Equally to moderately
1	Equally preferred

Example:

The service requester's quality preferences are:

- Availability (AV) is assigned by the service requester as two times more important than the Reputation (REP).
- Availability (AV) is assigned by the service requester as four times more important than the Price (P).
- Reputation is the same as important as Price.

The number of quality criteria, m=3. The requester specifies 3 preferences or judgments.Thus, a comparison matrix A from the equation [1] is formed:

$$A = \begin{array}{c} \\ AV \\ REP \\ P \end{array} \begin{array}{c} \begin{array}{ccc} AV & REP & P \end{array} \\ \begin{bmatrix} 1 & 2 & 4 \\ 1/2 & 1 & 1 \\ 1/4 & 1 & 1 \end{bmatrix} \end{array}$$

Step-2: Calculate the weight vector of quality criteria

The weights of quality criteria can be calculated from the matrix A by using equation (2).

$$w_i = \frac{1}{number\ of\ criteria}\left[\frac{preference\ of\ criterion\ i\ in\ column\ 1}{sum\ of\ the\ entries\ in\ column\ of\ criteria\ 1} + \frac{preference\ of\ criterion\ i\ in\ column\ 2}{sum\ of\ the\ entries\ in\ column\ of\ criteria\ 2} + \ldots + \frac{preference\ of\ criterion\ i\ in\ column\ m}{sum\ of\ the\ entries\ in\ column\ of\ criteria\ m}\right] \quad (2)$$

Example:

$$W(AV) = \frac{1}{3}\left[\frac{1}{1.75} + \frac{2}{4} + \frac{4}{6}\right] = 0.579$$

$$W(REP) = \frac{1}{3}\left[\frac{0.5}{1.75} + \frac{1}{4} + \frac{1}{6}\right] = 0.234$$

$$W(P) = \frac{1}{3}\left[\frac{0.25}{1.75} + \frac{1}{4} + \frac{1}{6}\right] = 0.187$$

The weight vector is:

$$W = [0.579 \quad 0.234 \quad 0.187]$$

Step-3: Calculate the Consistency Ratio (*CR)*

The Consistency Ratio (*CR*) measures the degree of consistency among the pair-wise judgements [11]. It can be calculated from equation (3) [12] . The Consistency Ratio (*CR*) of value 0.10 or less is considered acceptable and the requester judgement is consistent [10]. An acceptable consistency property helps to ensure decision-maker reliability in determining the priorities of a set of quality criteria.

$$CR = \frac{CI}{RI} \quad (3)$$

Where *CI* is the Consistency Index and *RI* is the Random Index. The *RI* value is selected from Table 2.

Table 2 Average Random Index (*RI*) [10]

Average random index (RI)										
Size of matrix	1	2	3	4	5	6	7	8	9	10
Random index	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

The Consistency Index (*CI*) is defined as [13], [14]:

$$CI = \frac{O - m}{m - 1} \quad (4)$$

Where O is the average of the row totals of the normalized matrix A divided by the weight vector

Example:

The Consistency Ratio (*CR)* is calculated from equations (3) and (4) as in the following.

1. Random Index *RI* for matrix A of size 3 is equal to 0.58, as given in Table 3.

2. Calculate O from the following:

   - Calculate the weighted sum matrix by the following:

$$\begin{bmatrix} 1 & 2 & 4 \\ 0.5 & 1 & 2 \\ 0.25 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 0.579 \\ 0.234 \\ 0.187 \end{bmatrix} = \begin{bmatrix} 1.795 \\ 0.711 \\ 0.566 \end{bmatrix}$$

   - Divide all the elements of the weighted sum matrices by their respective priority vector element to obtain:

$$\frac{1.795}{0.579} = 3.1 \quad , \quad \frac{0.711}{0.234} = 3.04 \quad , \quad \frac{0.566}{0.187} = 3.02$$

   - O can be obtained from the average of the above values:

$$O = \frac{3.1 + 3.04 + 3.02}{3} = 3.053$$

3. Calculate the Consistency Index CI from equation (4)

$$CI = \frac{O - m}{m - 1} = \frac{3.053 - 3}{3 - 1} = 0.0265$$

4. Calculate the Consistency Ratio (*CR)* from equation (3)

$$= \frac{}{} = \frac{0.0265}{0.58} = 0.046$$

The Consistency Ratio (CR) is equal to 0.046 which is less than 0.1, so the pair-wise requester's judgement is consistent and therefore the procedures will continue in order to select the best Web service.

Step-4: Normalize the proposed performance matrix

It is assumed that the performance matrix P, equation (5) is published by the service providers. The service providers publish their Web services with the same functional information but differ with their quality criteria values.

$$P = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1n} \\ p_{21} & p_{22} & \dots & p_{2n} \\ \dots & \dots & \dots & \dots \\ p_{m1} & p_{m2} & \dots & p_{mn} \end{bmatrix} \quad (5)$$

Since the criteria are measured in different measurement units, the performance matrix *P*, equation (5), should be converted into a non-dimensional one. This could be done as each element of *P* is normalized by the following calculation:

$$q_{ij} = \frac{p_{ij}}{\sqrt{\sum_{k=1}^{n} p_{ik}^2}} \quad (6)$$

This step produces a normalized performance matrix $Q = \{q_{ij}\}$.

The equation (6), considers only the increasing quality criteria that is the more the value the more benefit the service requester such as Availability and Reputation and it does not consider the decreasing quality criteria that is the more the value the less benefit the requester such as Price criterion. Further investigation required to consider the decreasing quality criteria as well the increasing criteria in the mathematical model.

Example:

Suppose that there are three Web services (n=3) have the same functional properties and published by different service providers, characterized by three quality criteria (m=3): C_1 =Availability, C_2 =Reputation and C_3 =Price. The values of the quality criteria are represented in a performance matrix P from the equation (5):

$$P = \begin{matrix} AV \\ REP \\ P \end{matrix} \begin{bmatrix} 95 & 99 & 95 \\ 4 & 3.5 & 3.5 \\ 38.37 & 30.27 & 38.38 \end{bmatrix}$$

The normalized performance matrix can be obtained from equation [8] as shown below:

$$\begin{bmatrix} 0.569 & 0.593 & 0.569 \\ 0.617 & 0.487 & 0.618 \end{bmatrix}$$

Step-5: Construct a weighted normalized performance matrix

The normalized values are then assigned weights with respect to their importance to the requester, given by the vector $w = \{ w_1, w_2, ..., w_m \}$.

When these weights are used in conjunction with the matrix of normalized values $Q = \{q_{ij}\}$, this produces the weighted normalized matrix $V = \{v_{ij}\}$, defined as $V = \{w_i q_{ij}\}$, or

$$V = \begin{bmatrix} w_1 q_{11} & w_1 q_{12} & \dots & w_1 q_{1n} \\ w_2 q_{21} & w_2 q_{22} & \dots & w_2 q_{2n} \\ \dots & \dots & \dots & \dots \\ w_m q_{m1} & w_m q_{m2} & \dots & w_m q_{mn} \end{bmatrix} \quad (7)$$

Example:

The weighted normalized performance matrix can be obtained from equation (7); $V = \{w_i q_{ij}\}$, where w_i is obtained from step-2, as shown below:

$$\begin{bmatrix} 0.329 & 0.343 & 0.329 \\ 0.115 & 0.091 & 0.116 \end{bmatrix}$$

Step-6: Calculate the relative distances

In this step each of the services is measured according to its closeness to the requester quality requirements. The relative Euclidean distances are calculated as follows:

$$E_j = \sum_{i=1}^{m} (v_{ij} - w_i r_i / \sqrt{\sum_{i=1}^{m} p_{ij}^2})^2 \quad (8)$$

Where *j=1,2,..., n* is the number of Web services.

Example:

Suppose that requester's quality requirements are $= (98, 3, 40)$ for the corresponding *Availability*, *Reputation* and *Price*. The values of the relative Euclidean distances, measuring the closeness between these requirements and the available services are obtained from equation (8):

$$_1 = 0.268\ ' \quad _2 = 0.239\ ' \quad _3 = 0.258$$

Step-7: Rank services in preference order

This is done by comparison of the values calculated in Step-6. Obviously, the Web service with smallest value $E^* = \min\{E_1, E_2, ..., E_n\}$ gives the closest match to the requester quality requirements and should be selected as the best one.

Example:

It is seen from the result of step-6 that the second Web service is the best one, since its Euclidean distance is smallest (0.239), compared to the distances of other services. So, the requester will select the second Web service.

If the requester's preferences are changed so that the weight vector is:

$$= [\ (\quad)\quad(\quad)\quad(\ )\ ] = [0.131\quad 0.677\quad 0.192]$$

Then the Euclidean distance will be:

$$_1 = 0.399\ ' \quad _2 = 0.398\ ' \quad _3 = 0.35$$

It is seen that the third Web service is the best for having the smallest Euclidean distance.

This example illustrates that the relative weight given to the quality criteria affects the final ranking of the service and depends on the requester preferences and therefore make certain quality criteria weigh more than others.

In the proposed quality-based Web service architecture (QWSA), it is considered to select more than one best service to be a more efficient approach; if one selected service failed, the others can be used instead.

III. Quality Matchmaking

Quality matchmaking is defined as a process that requires the quality matchmaker to match the quality inquiry to all the quality advertisements stored in the quality server's database, in order to find appropriate advertised services, which satisfy the quality requirements specified in the quality inquiry.

Different requesters may have different requirements and preferences regarding quality of Web service. For example, a requester may require to minimize the execution time while satisfying certain constraints in terms of price and reputation, while another requester may give more importance to the price than to the execution time [6]**.** Therefore, a quality matchmaking approach is needed to match quality requirements of requesters with the published quality specifications of providers in order to select the best service based on quality criteria constraints and preferences of the requesters.

The quality matchmaker is the core component in quality server. Every service request received by quality matchmaker will be matched with the service specifications that stored in the quality server database. If the match is successful, the quality matchmaker returns a ranked set of desired Web services and selects the appropriate service based on relevance quality criteria using mathematical technique.

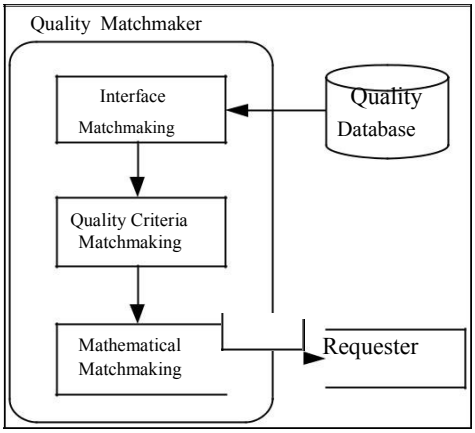


Figure 1 Quality Matchmaker

The quality matchmaker component includes the following sub-components (as shown in Figure 1)

- Interface matchmaking
- Quality criteria matchmaking
- Mathematical matchmaking

The roles of each sub-component are described in the following:

1) Interface Matchmaking

The interface matchmaking discovers the Web services which fitting functionality with the request requirements. Functionality means an action that either the service or the service requester can do [15]. This step finds all of the services matching the *interface* by using the operation called *find_tModel*() API on the UDDI registry. This step serves as an *interface matchmaking* filter and retrieves a list of all relevant description *tModels* for the services which have the same function. Once a set of tModels that match the specified requirements have been found, then a requester can find the corresponding services by using *find_service*() operation. This returns a list of all services that implement the description in the chosen *tModel* [16] then quality manager stores the result in the quality database.

The interface matchmaking is important but not sufficient to achieve requester satisfaction, because there are many services implement the same functional properties but have different non-functional (behaviour) properties and need to differentiate between them. Therefore, further matchmaking technique is needed regarding the quality criteria.

2) Quality Criteria Matchmaking

Quality criteria matchmaking compares the quality specifications with the quality requirements based on the quality descriptions of the services' behaviours. This step reduces or filters the returned list that is provided by the above interface matchmaking using the *quality criteria matchmaking* filter. The quality criteria matchmaking considers the structure of the quality criteria XML Schema [5]. The exact match occurs when the group quality criteria type and the sub-criteria type are same for both the quality requirements and the quality specifications.

Quality criteria matchmaking then uses the *quality value constraint matchmaking* filter in order to reduce the returned last list. The value of the required or preferred value of a certain quality sub-criteria type has to be within the range of the offered quality sub-criteria, and also the requested quality sub-criteria range is a subset of offered quality range. Further filtering needed to choose the optimum Web service from this list.

3) Mathematical Matchmaking

The mathematical matchmaking reduces the returned last list of services by using *mathematical matchmaking* filter in order to choose an optimum Web service.

The mathematical matchmaking ranks the services by calculating the distance between the required quality sub-criteria and the offered quality sub-criteria by using a mathematical model. The smallest distance means the best match and therefore the requester can select the best Web service. Once the services are ranked using Euclidean Distance technique, the requester needs to invoke the service by using *find_binding()* operation. This stage is explained in the following section.

IV. Quality Matchmaking Process

The quality matchmaking process (QMP) determines which Web service from the published Web services is the best service to be selected based on the requesters quality requirements and preferences. The matchmaking process is classified into two types:

- The first is the functional (interface) matchmaking that is used to search the UDDI for a Web service with the required functionality.
- The second is to use the quality criteria classification and a mathematical model to match the quality requirements against the quality specifications in the quality database.

The quality matchmaking process (QMP) has four algorithms or filters: Interface matchmaking (functional matchmaking), quality criteria type matchmaking (non-functional matchmaking), quality criteria value constraint matchmaking and mathematical matchmaking. Each of these algorithms or filters narrows a set of matchmaking candidates with respect to a given filter

criterion. These four algorithms are illustrated below with an example using Amazon E-Commerce Service (ECS) case study [17] .

Step -1: Interface Matchmaking Algorithm:

This step finds all of the matching services that only consider the published Web services matching the required interface. Figure 1 shows a flow chart of an interface matchmaking algorithm that matches the advertised functional specifications in the Web services database with the functional requirements and keeps the result in an *iList* array.

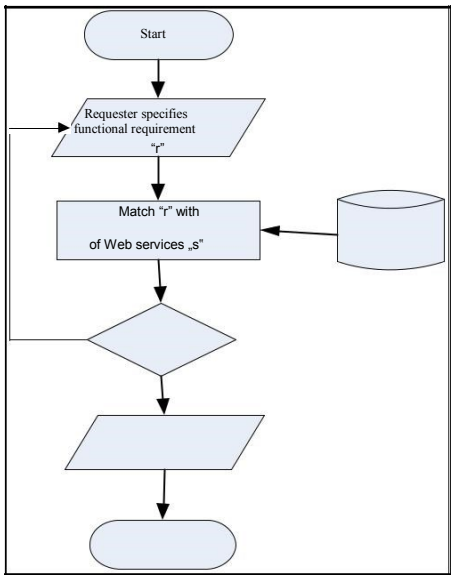
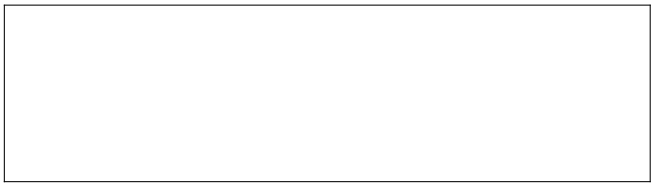


Figure 1 Interface Matchmaking Flow Chart

Example:



Listing 1 REST Request

The service requester sends his functional requirements to the quality matchmaker. The quality matchmaker sends REST request to the ECS database as shown in Listing 1. In ECS there are two types of request REST (XML over HTTP) and SOAP request.

The interface description as shown in Listing 1 includes the following:

- Operation request *ItemSearch*. Amazon E-Commerce Service
- (ECS) [17] provides two types of inquiries: search and lookup request.
- SearchIndex *Books*. ECS provides several search indexes: Books, Music, Computer, etc.
- Title *Web Services.* Title is a parameter to the *ItemSearch* operation.
- ResponseGroup: specifies the type of the retrieved information.

The interface matchmaking steps are:

- The quality matchmaker first searches the ECS database using I*temSearch* operation. The matchmaker matches the keyword *Web Services* with the offered books within the *Books* category.
- The matchmaker returns a large list *iList* of matched books includes *Web Services* keyword.

Step-2: Quality Criteria Type Matchmaking Algorithm:

This step is based on quality criteria classification structure. Figure 2 shows a flow chart of a quality criteria and sub-criteria matchmaking algorithm. The service requester selects the quality criteria and sub-criteria. The required criteria type (such as Performance, failure Probability, Trustworthiness, and/or Cost) and the sub-criteria type (such as Response Time, Availability, reputation, etc.) are matched with the advertised criteria and sub-criteria type, which are saved in the returned list *iList* in step-1. If both the required and advertised criteria and sub-criteria type are same, then the result is saved in an *sqList*[] array. This paper for simplicity assumes that the criteria and sub-criteria type of the advertised services are always similar.

Example:

The above result which stored in *iLis*t is filtered by using quality criteria type matchmaking algorithm. The matchmaker returns a list *sqList* of services contains the following sub-criteria: Availability, Reputation, and Service Price.

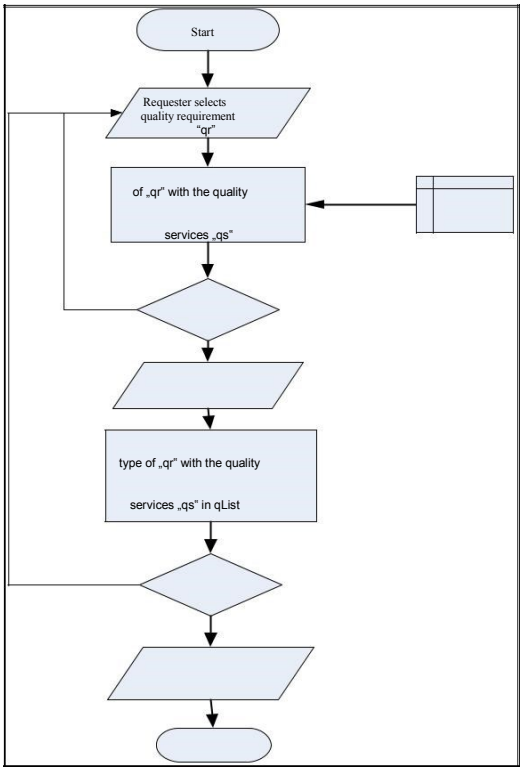


Figure 2 Quality Type Matchmaking Flow Chart

Step-3: Quality Criteria Value Matchmaking Algorithm:

This step is based on the quality sub-criteria level (High, Medium, or Low) that the requester specifies. Each quality level has a preferred value. The returned list *sqList* from step-2 is further filtered by using quality criteria value matchmaking algorithm as shown in Figure 3. The following rule must be satisfied in order to save the result in *qvList* array list: qlr<=qls

That is, the required quality sub-criteria value must be less than or equal the advertised quality sub-criteria value.

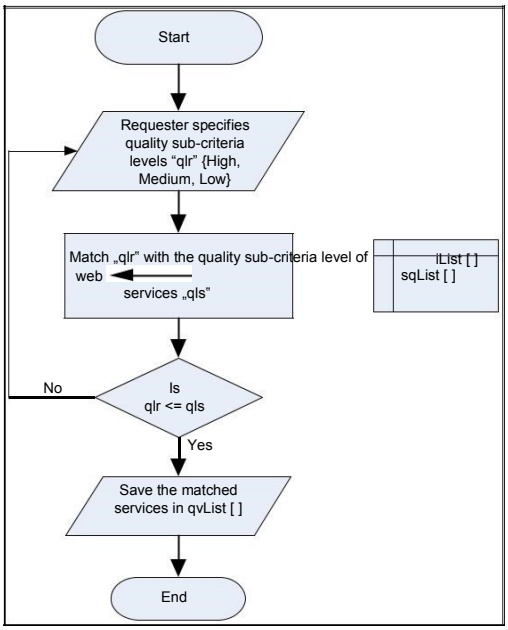


Figure 3 Quality Value Matchmaking Flow Chart

Example:

The returned result which stored in *sqList* is further filtered by using quality sub-criteria value constraints matchmaking. The matchmaker returns a list of services *qvList* which their offered quality values are within the range of the required values. The ranges of the required quality values are related to the required quality level parameter *qlevel* (High, Medium, or Low) as shown in Figure 5. The query is shown in Listing 2.

```
SELECT Availability, Reputation, ServicePrice
FROM QualityDatabase
WHERE QualityDatabase.Availability= ´ + L J K ´  $ 1 '
QualityDatabase.Reputation= ´ 0 H G L X P ´  $ 1 '
QualityDatabase.ServicePrice= ´ 0 H G L X P ´
```

Listing 2 SQL Query

Quality Requirement Description

Operation=ItemSearch
SearchIndex=Books
Title=Web Services
Availability= qlevel: High
Min: 90
Max: 99
Unit: Percentage
Weight: 0.579
Reputation= qlevel: Medium
Min: 2.5
Max: 4
Unit: None
Weight: 0.234
ServicePrice= qlevel: Medium
Min: 30
Max: 60
Unit: Pound
Weight: 0.187

Figure 4 Example of Quality Requirement provided by Service Requester

The quality database is the database in the quality server. Figure 5 shows the result of quality value matchmaking algorithm. It shows different providers providing services with the same functional specifications but different in its quality specifications.

Quality Specifications Description

Service Provider1	Service Provider2	Service Provider3
Service1 Specification:	**Service1 Specification:**	**Service1 Specification:**
Title= Understanding Web Service:XML, WSDL, SOAP, and UDDI	Title= Understanding Web Service:XML, WSDL, SOAP, and UDDI	Title= Understanding Web Service:XML, WSDL, SOAP, and UDDI
Availability=98	Availability=90	Availability=99
Reputation=4	Reputation=4.8	Reputation=3.5
ServicePrice=29.07	ServicePrice=39.69	ServicePrice=30.27
Service2 Specification:	**Service2 Specification:**	**Service2 Specification:**
Title=Web Services Security	Title=Web Services Security	Title=Web Services Security
Availability=90	Availability=95	Availability=90
Reputation=4	Reputation=4.8	Reputation=3.5
ServicePrice=26.44	ServicePrice=42.94	ServicePrice=28.47
Service3 Specification:	**Service3 Specification:**	**Service3 Specification:**
Title=J2EE Web Services	Title=J2EE Web Services	Title=J2EE Web Services
Availability=95	Availability=99	Availability=95
Reputation=4	Reputation=4.8	Reputation=3.5
ServicePrice=38.37	ServicePrice=45.72	ServicePrice=38.38

Figure 5 Example of Quality Specifications Description provided by Service Providers

The result is organised in the following matrix:

$$P = \begin{matrix} AV \\ REP \\ \end{matrix} \begin{bmatrix} 95 & 99 & 95 \\ 4 & 3.5 & 3.5 \\ 38.37 & 30.27 & 38.38 \end{bmatrix}$$

The first row is related to sub-criterion Availability (AV), the second row is related to Reputation (REP), the third row is related to Service Price (P).

The first column is related to book with title ―J2EE Web Services‖ which provided by provider 1 (see Figure 5), the second column is related to book title ―Understanding Web Service: XML, WSDL, SOAP, and UDDI‖ which provided by provider 3, the third column is related to book title ―J2EE Web Services‖ which provided by provider 3.

Step-4: Mathematical Matchmaking Algorithm

This step is based upon a mathematical model that explained in Section II. This step is the most important step in the quality matchmaking process (QMP) (see section IV). The mathematical matchmaking algorithm selects the best Web service from the last list *qvList* from step-3 as shown in Figure 6. The service requester specifies the selected quality criteria and sub-criteria preferences. The weight of the quality criteria and sub-criteria is calculated using Analytical Hierarchy Process. Then the consistency ratio (CR) must be less than 0.1 to continue the process. Then the Euclidean distance measures the distance between the requester's quality requirements and the provider's quality specifications of the services that are saved in *qvList*[] array from step-3. The service associated with a minimum distance is the best service to select. The AHP and Euclidean distance are explained in Section II.

Example:

The mathematical technique (Analytical Hierarchy process and Euclidean Distance) is used to measure the distance between the quality requirements and the quality specifications. The minimum distance calculated will be the best service to select. After using the mathematical technique the final result are:

The distance of the book title ―J2EE Web Services‖ which provided by provider 1 is: 0.268.

The distance of the book title ―Understanding Web Service: XML, WSDL, SOAP, and UDDI‖ which provided by provider 3 is: 0.239.

The distance of the book title ―J2EE Web Services‖ which provided by provider 3 is: 0.258.

From the above result the minimum distance is 0.239 which is related to the book title ―Understanding Web Service: XML, WSDL, SOAP, and UDDI‖ and provided by provider 3, so this is the best book which the requester can select to buy. It is noticed from the result that the book with highest Availability value is selected and it is reasonable because the requester specifies the quality level *qlevel* for the Availability sub-criterion to High, whereas for Reputation and Service Price for Medium, this affect to the weight priority of the Availability which is the highest priority (0.579) and therefore affect the book selection.

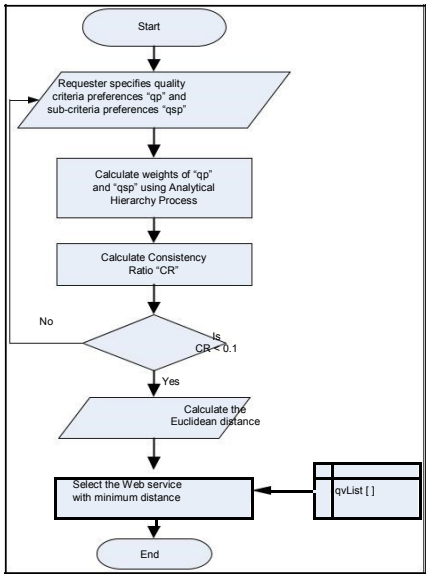


Figure 6 Quality Mathematical Matchmaking Flow Chart

V. Implementing Mathematical Matchmaking Algorithm

The mathematical matchmaking algorithm has been implemented by developing a *Utilities* class using Visual Studio .NET 2005.

Utilities class contains the *Matrix* class and methods such as: *FillMatrix*(), *CalculateWeights*(), *ConsistencyRatio*() and *EuclideanDistance*(). The matrix class and the methods are described below.

***Matrix* class**

Matrix class is used to create matrix instances. The matrix is a multidimensional array is shown in Figure 8.

```
public class Matrix
        {
                double[,] matrix;
                int numberOfRows, numberOfColumns; public
                Matrix(int rows, int columns) {

                        numberOfRows = rows;
                        numberOfColumns = columns;
                        matrix = new double[rows, columns];
                }

                //    Constructor to initialize the data in the matrix public double this[int i,
                int j]
                {
                        set { matrix[i,j] = value; } get { return
                        matrix[i,j]; }
                }

                //    Return number of rows in the matrix
                public int Rows
                {
                        get { return numberOfRows; }
                }

                //    Return number of columns in the matrix public int
                Columns
                {
                        get { return numberOfColumns; }
                }
        }
```

Figure 8 Matrix Class

FillMatrix() method

FillMatrix() method as shown in Figure 9 is used to construct pair-wise comparison matrix A that is based on the service requester's quality preferences.
The input parameters to *FillMatrix*() method are the requester's quality preferences. The output of the *FillMatrix*() method is the pair-wise comparison matrix A.

The number of the columns and the rows of matrix A, is equal to the number of quality criteria (i.e. Trustworthiness), or sub-criteria (i.e. reputation).

```
//fillMatrix0 method construct pair-wise comparison matrix based on the service //
        requester's criteria and sub-criteria preferences

                public    void fillMatrix0(Matrix A, double[] arrValue)
               {
                 //if the service requester selects only one quality criteria
                        if(A.Rows==1)
                        {
                                for (int i=0;i<A.Rows;i++)
                                {
                                        for(int j=0;j<A.Rows;j++)
                                        {
                                                A[i,j]=1;
                                                A[j,i]=1;
                                        }
                                }
                        }
    //if the service requester selects more than one quality criteria else
                        if(A.Rows>1)
                        {
                                for (int i=0;i<A.Rows-1;i++)
                                {
                                        for(int j=i+1;j<A.Rows;j++)
                                        {
                                                double nextVal = getNextValue(arrValue);
                                                if(nextVal != -1)
                                                {
                                                        A[i,j]=nextVal;
                                                    A[j,i]=1/nextVal;
                                                        A[i,i]=1;
                                                        A[j,j]=1;
                                                }
                                        }
                                }
                        }
                }
```

Figure 9 FillMatrix() Method

CalculateWeights() method

CalculateWeights() method as shown in Figure 10 is used to calculate the criteria and the sub-criteria weights from the pair-wise comparison matrix A. This method is explained in Section II.

The input parameters to *CalculateWeights*() method are the matrix A and the number of selected criteria. The output of the *CalculateWeights*() method is an array this contains the weights of the selected quality criteria.

```
//    calculateWeights() method calculates the criteria and sub-criteria weights from pair-wise
comparison matrix
public  double[] calculateWeights(Matrix MatrixA, int criteriaNumber)
                {
                //calculate the sum of each column in MatrixA
                        criteriaNumber= MatrixA.Rows;
                        double [] Sum = new double[criteriaNumber];
                        for(int j=0; j<criteriaNumber; j++)
                        {
                                for(int i=0; i<criteriaNumber; i++)
                                {
                                        Sum[j]=Sum[j]+MatrixA[i,j];
                                }
                        }

                // create the normalized matrix Normalised
                //by dividing each entry in the matrix by its column sum
                        Matrix Normalised = new Matrix(criteriaNumber,criteriaNumber); for(int j=0;
                        j<criteriaNumber; j++) {

                                for(int i=0; i<criteriaNumber; i++)
                                {
                                        Normalised [i,j]=MatrixA[i,j]/Sum[j];
                                }
                        }

                //Calculate the weight of each criteria
                //which is equal to the avarage of its corresponding row double []
                        WeightCriteria = new double[criteriaNumber]; double sumOfRow = 0;

                        for(int i=0; i<criteriaNumber; i++)
                        {
                                for(int j=0; j<criteriaNumber; j++)
                                {
                                        sumOfRow=sumOfRow+Normalised[i,j];
                                        WeightCriteria[i]=sumOfRow/criteriaNumber;
                                }
                                sumOfRow=0;
                        }
                        return WeightCriteria;
                }
```

Figure 10 CalculateWeight() Method

ConsistencyRatio() method

ConsistencyRatio() method as shown in Figure 11, is used to calculate Consistency Ratio (*CR*). The CR measures the degree of consistency of the selected preferences values of the quality criteria that considered as a condition for allowing the service requester to continue the selection procedures or to specify new quality preferences values. This method is explained in Section II.

The input parameters to *ConsistencyRatio*() method are the matrix A, the number of selected criteria and the weights array. The output of the *ConsistencyRatio*() method is the Consistency Ratio (CR) value.

```
//ConsistencyRatio() method calculated the Consistenct Ratio (CR)
                public double ConsistencyRatio (Matrix A, double [] weight, int criteriaNumber)
                {
                double consistencyIndex;
                        double consistencyRatio;
                double randomIndex=1;
                        double sum=0;
                        double weightSum=0;
                        double eigenMax;
                        double [] eigenValue=new double[criteriaNumber];
//    the values of Random Index (RI)for differrent number of criteria selected
//    3<=RI<=10
                        if (criteriaNumber==3)
                        {
                                randomIndex=0.58;
                        }
                        if (criteriaNumber==4)
                        {
                                randomIndex=0.9;
                        }
                        if (criteriaNumber==5)
                        {
                                randomIndex=1.12;
                        }
                        if (criteriaNumber==6)
                        {
                                randomIndex=1.24;
                        }
                        if (criteriaNumber==7)
                        {
                                randomIndex=1.32;
                        }
                        if (criteriaNumber==8)
                        {
                                randomIndex=1.41;
                        }
                        if (criteriaNumber==9)
                        {
                                randomIndex=1.45;
                        }
                        if (criteriaNumber==10)
                        {
                                randomIndex=1.49;
                        }
        //calculate the eigenvalue max
                        for(int i=0; i<criteriaNumber; i++)
                        {
                                for (int j=0; j<criteriaNumber; j++)
                                {
                                        weightSum=weightSum+weight[j]*A[i,j];
                                }
                                eigenValue[i]=weightSum/weight[i];
                                weightSum=0;
                        }
                        for(int k=0; k<criteriaNumber; k++)
                        {
                                sum=sum+eigenValue[k];
                        }
                        eigenMax=sum/criteriaNumber;
    //calculate the Consistency Index (CI)
                        consistencyIndex=(eigenMax-criteriaNumber)/(criteriaNumber-1);
        //calculate     the Consistency Ratio (CR)
                        consistencyRatio=consistencyIndex/randomIndex; return
                        consistencyRatio;
                }
```

Figure 11 ConsistencyRatio() Method

EuclideanDistance() method

EuclideanDistance() method as shown in Figure 12, is used to calculate the Euclidean distance of the advertised Web services. The service with the smallest distance is the best one that the service requester can select it. This method is explained in Section II.

The input parameters to *EuclideanDistance*() method are the performance matrix P; this contains the advertised services, the number of selected criteria, the weights array and an array of the quality requirement values. The output of the *EuclideanDistance*() method is an array of the Euclidean distance values for all the advertised services in matrix P.

```
//    EuclideanDistance() method calculates the Euclidean distance for each service in the performance
matrix
            public double[]EuclideanDistance(Matrix P, int subCriteriaNumber, int serviceNumber, double[]
Weight,double []requirement)
            {
                  subCriteriaNumber=P.Rows;
                  serviceNumber=P.Columns;
                  double sum=0;
                  double[] Sqrt=new double[subCriteriaNumber];
                  for(int i=0; i<subCriteriaNumber;i++)
                  {for(int j=0; j<serviceNumber; j++)
                        {
                              sum=sum+P[i,j]*P[i,j];
                        }
                     Sqrt[i]=Math.Sqrt(sum);
                        sum=0;
                  }
      // calculate the normalized performance matrix
                  Matrix PNormalised = new Matrix(subCriteriaNumber,serviceNumber);
                  for(int i=0; i<subCriteriaNumber; i++)
                  {
                        for(int j=0; j<serviceNumber; j++)
                        {
                              PNormalised [i,j]=P[i,j]/Sqrt[i];
                        }
                  }
      // create V matrix by multiplying weight vector with the normalized performance matrix

                  Matrix V =new Matrix(subCriteriaNumber, serviceNumber);
                  for(int i=0; i<subCriteriaNumber; i++)
                  {
                        for(int j=0; j<serviceNumber;j++)
                        {
                              V[i,j]=Weight[i]*PNormalised[i,j];
                        }
                  }
   //multiply the weight vector with requirement value vector
                  double[] wr=new double[subCriteriaNumber];
                  for(int i=0; i<subCriteriaNumber;i++)
                  {
                        wr[i]=Weight[i]*requirement[i];
                  }
                  double[] SqrtC=new double[serviceNumber];
                  for(int j=0; j<serviceNumber; j++)
                  {
                        for(int i=0; i<subCriteriaNumber; i++)
                        {
                              sum=sum+P[i,j]*P[i,j];
                        }
                     SqrtC[j]=Math.Sqrt(sum);
                        sum=0;
                     }
   //calculate the Euclidean distance
                  double[] EucDistance=new double[serviceNumber];
                  double finalSum=0;
                  for(int j=0; j<serviceNumber; j++)
                  {
                        for(int i=0; i<subCriteriaNumber; i++)
                        {
                    finalSum = finalSum +(V[i,j]-(wr[i]/SqrtC[j]))*(V[i,j]-(wr[i]/SqrtC[j]));
                        }
                     EucDistance[j]=Math.Sqrt(finalSum);
                        finalSum=0;
                  }
               return EucDistance;
            }
```

Figure 12 EuclideanDistance() Method

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed the role of the quality matchmaker component, which is the core component in the proposed quality-based Web service architecture (QWSA). The quality matchmaker introduces four algorithms or filters: interface matching, quality criteria matchmaking, quality value constraints matching, and mathematical matchmaking. These four algorithms use the quality matchmaker sub-components to implement their roles. The quality matchmaker has three sub-components which are: interface matchmaking, quality criteria matchmaking and mathematical matchmaking.

A quality matchmaking process (QMP) is introduced to demonstrate the above four algorithms and to select the best Web service. The last step in the matchmaking process is a mathematical matchmaking algorithm. It is the most important step that uses a mathematical model in order to select the best candidates Web service based on requester's quality requirements and preferences. Two techniques are used in a mathematical model: Analytical Hierarchy Process (AHP) and Euclidean distance.

QMP is illustrated by an example using Amazon E-Commerce Service (AEC) case study. This example shows how the service selection is affected by two factors: the criteria weights and the quality requirements values

The proposed quality matchmaking process (QMP) has been derived with the assumption that the query, which is sent by the service requester, is volatile that is no new services will be added to UDDI and no changes to the quality criteria values for these services. These limitations will be further investigated by adapting the requesters to any changes in the quality criteria during a long time query.

VII. REFERENCES

[1] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," 24 June 2003. Available at :http://www.w3c.org/TR/SOAP12-part1.

[2] E. Christensen, F. Curbea, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," March 2001. Available at: http://www.w3.org/TR/wsdl.

[3] A. Manes, "Web Services Standardization: UDDI," 19 September 2003. Available at: http://www.uddi.org/news.html.

[4] W3C Working Group, "Web Services Architecture," Feb. 2004.

[5] A. Eleyan and L. Zhao, "Extendind WSDL and UDDI with Quality Service Selection Criteria," in *The 3nd International Symposium on Web Services* Zayed University, Dubai, U.A.E, 2010.

[6] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IEEE Transactions on Software Engineering,* vol. 30, pp. 311 - 327, 2004.

[7] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS computation and policing in dynamic web service selection," in *International World Wide Web Conference*, New York, NY, USA, 2004.

[8] P. Fedosseev, "Composition of Web Services and QoS Aspects," Seminar: Data Communication and Distributed Systems in the WS 2003/2004.

[9] L. Taher, H. El Khatib, and R. Basha, "A Framework and QoS Matchmaking Algorithm for Dynamic Web Services Selection," in *Second International Conference on Innovations in Information Technology (IIT'05)* Dubai, UAE, 2005.

[10] T. L. Saaty, "How to make a decision: The Analytic Hierarchy Process," *European Journal of Operational Research,* vol. 48, pp. 9-26, 1990.

[11] H. Ye, B. Kerherve, and G. V. Bochmann, "QoS-based Distributed Query Processing," *Ingénierie des Systèmes d'Information (RSTI série ISI),* vol. 9, 2004.

[12] M. Hajeeh and A. Al-Othman, "Application of the analytical hierarchy process in the selection of desalination plants," *Desalination,* vol. 174, pp. 97-108, 2005.

[13] L. Taher, R. Basha, and H. El Khatib, "Establishing Association between QoS Properties in Service Oriented Architecture," in *Proceedings of the IEEE International Conference on Next Generation Web Services Practices (NWeSP'05)*, 2005.

[14] L. Taher, H. Khatib, and R. Basha, "A Framework and QoS Matchmaking Algorithm for Dynamic Web Services Selection," in *The Second International Conference on Innovations in Information Technology (IIT'05)*, 2005.

[15] S. Andreozzi, D. Montesi, and R. Moretti, "Web Services Quality," in *Conference on Computer, Communication and Control Technologies (CCCT03)*, Orlando, 31 July - 2 August 2003.

[16] J. Colgrave, R. Akkiraju, and R. Goodwin, "External matching in UDDI," in *IEEE International Conference on Web Services (ICWS'04)*, San Diego, California, June 2004.

[17] "Amazon Web Services," Available at: http://amazon.com/webservices.