ISSN 2007-9737

Open Framework for Web Service Selection Using Multimodal and Configurable Techniques

Oscar Cabrera[1], Marc Oriol[1], Xavier Franch[1], Jordi Marco[1], Lidia López[1], Olivia Graciela Fragoso Díaz[2], and René Santaolaya[2]

[1] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

[2] Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), Morelos, Mexico

{ocabrera, moriol, franch, llopez}@essi.upc.edu, jmarco@lsi.upc.edu, {ofragoso, rene}@cenidet.edu.mx

Abstract. Services as part of our daily life represent an important means to deliver value to their consumers and have a great economic impact for organizations. The service consumption and their exponential proliferation show the importance and acceptance by their customers. In this sense, it is possible to predict that the infrastructure of future cities will be supported by different kind of services, such as smart city services, open data services, as well as common services (e.g., e-mail services), etc. Nowadays a large percentage of services are provided on the web and are commonly called web services (WSs). This kind of services has become one of the most used technologies in software systems. Among the challenges when integrating web services in a given system, requirements-driven selection occupies a prominent place. A comprehensive selection process needs to check compliance of Non-Functional Requirements (NFRs) which can be assessed by analyzing the Quality of Service (QoS). In this paper, we describe a framework called *WeSSQoS* that aims at ranking available WSs based on the comparison of their QoS and the stated NFRs. The framework is designed as an open Service Oriented Architecture (SOA) that hosts a configurable portfolio of normalization procedures and ranking algorithms which can be selected by users when starting a selection process. The QoS data from WSs can be obtained either from a static, WSDL-like description or dynamically through monitoring techniques. *WeSSQoS* is designed to work over multiple WS repositories and QoS sources. The impact of having a portfolio of different normalization and ranking algorithms is illustrated with an example.

Keywords. Web service (WS), web service selection, service oriented architecture (SOA), quality of service (QoS), non-functional requirement (NFR), service level agreement (SLA), ranking services.

1 Introduction

In today´s world, there are different kinds of services created to facilitate the life of citizens in their daily tasks. These services have been developed to solve different needs according to certain requirements of different human desires. As a result, an enormous explosion in offering services has occurred. In fact, it can be observed that for a given need, a plethora of services can be found. In addition, according to [1] there is a growth in consumer services driven by various social, economic, and technological factors (e.g., a demand for social services, the size and role of the public sector, complexity of work environments, etc.).

A generic definition of a service is provided by the Office of Government Commerce (OGC) in its ITIL standard as follows [1]: *"A service is a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks"*.

The OGC considers that the outcomes mentioned are possible through the performance of different tasks and are limited by the presence of certain constraints. In this sense, the presented paper is focused on quality constraints that characterize services. Specifically, web services (WSs), since a large amount of services are being provided using this technology.

WSs integrate a set of protocols and standards for data interchange among software applications developed in different programming environments and languages, and executed in different platforms. This interoperability is provided mainly by the following open standards: XML, SOAP, HTTP, WSDL, and other web-related standards [2].

WSs have become a useful technology to implement any kind of software, providing greater interoperability and scalability. This success has triggered the emergence of a huge WSs marketplace. Consequently, for a given functionality we may find a large set of WSs that can be selected in several ways. This proliferation of WS increments the chances to find existing software that satisfies the stated needs, but at the same time raises new problems and challenges. Among them, there is an increasing need for selecting the most appropriate WS in a given context of usage [3]. Usually this problem is studied in relation with the requirements elicited from the stakeholders. In other words, the goal is to select the WS that "better" satisfies the stakeholder requirements.

We consider here the classical distinction among functional and non-functional requirements [4]. With respect to functional requirements, it is necessary to validate that a WS fulfills the functionality expected by the stakeholders. On the other hand, non-functional requirements (NFRs) refer to the Quality of Service (QoS) that a given WS offers, i.e., behavioral and non-behavioral characteristics that the WS exhibits for offering a given functionality: cost, response time, availability, etc. Usually, NFRs are expressed in terms of conditions over the QoS in a document named Service Level Agreement (SLA). Therefore, we can assess if a WS *w* satisfies an NFR *r* by checking if the QoS of *w* satisfies the clauses from the SLA that refer to the concepts inherent to *r*.

Given this context, our work proposes a framework for ranking a set of WSs that belong to a certain software domain. We assume that the functional requirements are used to determine this software domain, therefore our framework focuses on the ranking based on the satisfaction of NFRs.

The main goals that we aim to address in this paper are: how NFRs are expressed; what measure of the satisfaction of an NFR in a given WS is; how these individual measures are combined in order to rank the WS according to a set of NFRs; how the QoS of a WS may be obtained; where the WSs are obtained from; and what the value obtained by combining different normalization procedures and ranking algorithms to select WSs is. To attain these goals we have designed *WeSSQoS* (Web Service Selection based on Quality of Service), a framework for selecting WSs based on their QoS and NFRs. *WeSSQoS* proposes an open Service Oriented Architecture (SOA) that is able to manage several ranking algorithms and normalization procedures for computing the adequacy of a WS with respect to NFRs. These NFRs are expressed by means of formulae stated over QoS attributes (i.e., SLA clauses) coming from the quality model proposed in an earlier work [5]. NFR are classified as mandatory and optional, and this information may be used for ranking the results. *WeSSQoS* is designed to work over several WS repositories that eventually can be built using different technologies. In order to get the behavior of the accessible WSs with respect to the selection criteria, it is possible to use either the description of the QoS (if included in the WS definition), or the results of WS monitoring (obtaining then the real, updated QoS of the WS). In this sense, we share the vision of [6] that proposes to define a priori only static attributes like cost, whilst dynamic attributes like response time or availability should be obtained through monitoring.

The rest of the paper is organized as follows. In Section 2 a review of some similar frameworks is provided. Then, Section 3 describes the proposed WS selection process. Section 4 introduces normalization procedures and ranking algorithms. Section 5 describes the framework architecture proposed. Sections 6 and 7 describe a prototype and provide some validation. Finally, Section 8 presents conclusions and future work.

2 Related Work

In the academic research, different frameworks for ranking and selecting WSs according to their

QoS have been proposed. In Table 1 we show a representative sample of these proposals, including our *WeSSQoS* framework, comparing them according to the following criteria:

a. Architectural style: architecture in which the framework has been developed. We find Component Based Architectures (CBA), Service Oriented Architectures (SOA) and a combination of both. We represented each of the styles by using C, S, and CS, respectively. It is worth noting that adopting SOA allows integrating heterogeneous systems more easily.

b. Attributes: quality attributes considered in those systems. In some cases, a small predefined set of quality attributes is used, whereas other frameworks allow the usage of arbitrary ones (although they may validate the proposal with a given set). We represent the value of this criterion by using the amount of attributes defined as dynamic (d) or as static (s). In case of configurable attributes, i.e., the possibility of adding new attributes, we use an asterisk (*).

c. QoS data source specifies if quality data are declared in the service description (represented by using S) or for dynamic quality

Table 1. Comparative table of frameworks

Proposal	(a)	(b)	(c)	(d)	(e)	(f)	(g)
E. Al-Masri et al. [6]	C	6d 3s	SM	X	X	✓	✓
T. Yu et al. [7, 8]	C	4d 1s	S	X	✓	X	X
X. Wang et al. [9]	-	*1d5s	S	X	X	X	X
D. D' Mello et al. [10]	CS	*3d2s	SM	X	X	X	X
H. Wang et al. [11]	C	6s	SM	X	X	X	X
P. Wang et al. [12]	-	*6d	S	X	X	X	X
R. Mohanty et al. [13]	-	9s	S	X	✓	X	X
Q. Tao et al. [14]	C	6s	SM	X	X	X	X
H. Cai et al. [15]	CS	*0s	SM	X	X	X	X
L. Sha et al. [16]	CS	7s	SM	X	X	X	X
M. Alrifai et al. [17]	C	0s	SM	X	X	X	X
A. Huang et al. [18]	-	0s	S	X	X	X	X
C. Lin et al [19]	C	0s	S	X	X	X	X
Z. Gao et al. [20]	-	5s	S	X	X	X	X
WeSSQoS	CS	*9d1s	SM	* ✓	* ✓	*✓	✓

attributes, if their value is obtained through monitoring (represented by using M). In cases where the proposals provide both kinds of sources we represent the value by using SM.

d. Multinormprocedure specifies if the framework is able to work with more than one normalization procedure in order to obtain the normalized QoS data about WSs and stakeholders. We represent the value of this criterion by using yes (✓) or no (X). In case of proposals which allow adding new procedures, we use an asterisk (*).

e. Multialgorithm specifies if the framework is able to work with more than one selection algorithm. We represent the value of this criterion by using yes (✓) or no (X). In case of proposals which allow adding new algorithms, we use an asterisk (*).

f. Multirepository specifies if the framework is able to obtain data from different repositories and combine information to extend the number of services and quality attributes to evaluate. We represent the value of this criterion by using yes (✓) or no (X). In case of proposals which allow adding new repositories, we use an asterisk (*).

g. Prototype *available* specifies if the framework is available to be used for the research community. We represent the value of this criterion by using yes (✓) or no (X).

As a result of the previous evaluation we identified different gaps, such as a lack of frameworks with the capability to retrieve a list of web services from different sources. As far as we know, the only framework that fulfills this criterion is provided by Al-Masri *et al*. [6], whose framework obtains a list of WSs from several sources (UDDIs, ebXMLs, search engines, and service portals). However, it does not specify the method to combine the services data when different sources have the same service with different QoS data: cost, brand reputation, etc.

Another important gap is a lack of frameworks with the capability to reuse existing normalization procedures and selection algorithms, which would allow assessing results obtained from different proposals. In fact, only QCWS [7, 8] offers the capability of *multialgorithm*. However, since it is not a SOA, it does not allow adding external algorithms in an easy manner.

Regarding the criterion of *prototype available*, we identified that although in most of the proposals a prototype is being described and even some of them have a web page (e.g., [7, 8]), there is not a framework available. In fact, we have only found a tool available from Al-Masri *et al*. [6].

3 The Proposed Web Service Selection Process

Figure 1 shows the proposed web service selection process with the following inputs and selection phases:

Inputs:

- WSlist, a QoS matrix of size $k \times n$, where (w_1, …, w_k) are the candidate WS and (q_1, …, q_n) are the quality attributes referred in the NFRs. WSlist[*i, j*] stands for the value of the quality attribute q_j in the WS w_i.
- lreqs, a NFR vector of size *n*, where lreqs[*i*] specifies (1) the value that is required for the attribute q_i, (2) a Boolean value that indicates if the attribute's value is to be minimized or maximized, and (3) another Boolean value that indicates if the required attribute's value is mandatory or not. A value is mandatory when it cannot be higher than the required threshold when it has to be minimized, or cannot be lower than the threshold when it has to be maximized, e.g., a NFR may state to minimize the cost mandatorily with a maximum of 100 euros per month.

Selection phases:

- Normalization. This phase has the purpose of integrating the heterogeneous QoS attributes' values over which decision-making in the WSs selection problem relies. Both inputs WSlist and lreqs must be normalized to compensate the different measurement units of the different QoS values by projecting them onto a normalized interval. Interval boundaries are established by the normalization procedure used. Details of the different normalization procedures are

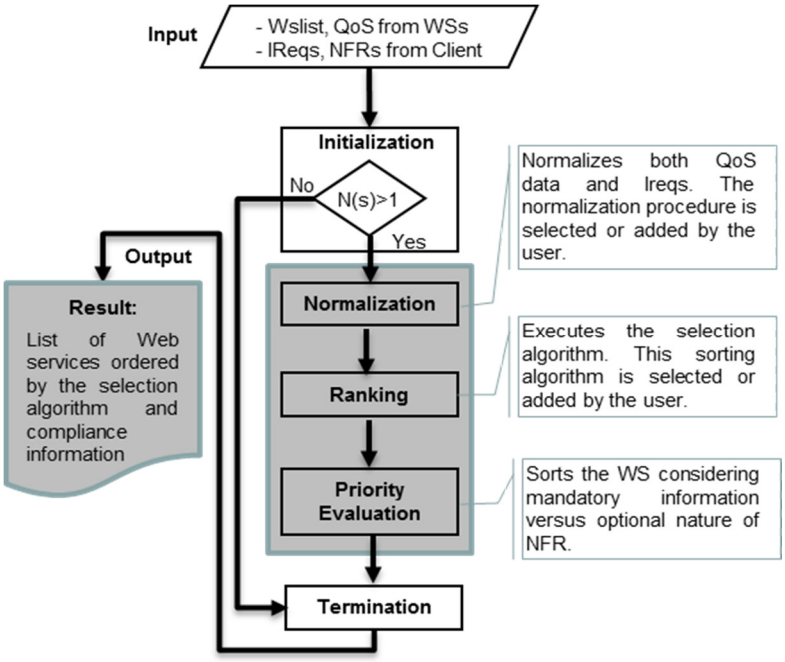

Fig. 1. Flow diagram for the web services selection process

described in the next section. The result of this phase is the normalized structures denoted by WSlistN and IreqsN.

- Ranking. Starting from the normalized data in the previous stage, a ranking algorithm can be applied with the goal of computing a similarity measure between the NFR (IreqsN) and the QoS of each service (WSlistN). This algorithm may be any of the commonly employed in Vector Space Models (VSM) to evaluate the similarity between two objects described by vectors [21]. For example, the Euclidian Distance algorithm looks for the shortest distances between the QoS of each candidate WS and the user NFR. As a result, we obtain the values of the algorithm and the WSs ranked according to them. Next section describes different ranking algorithms.
- Priority evaluation. In this phase two main types of WSs ranking are carried out, one by the number of mandatory requirements that services fulfill and one by the selection algorithm used.

4 Normalization Procedures and Ranking Algorithms

One of the main characteristics of the proposed framework architecture is that it supports the coexistence of normalization procedures and ranking algorithms offered as services.

4.1 Normalization Procedures

The normalization service that *WeSSQoS* currently offers has four normalization procedures (see equations 1 to 4). Nevertheless, as mentioned before, users can extend it by providing their own normalization procedures. In this sense, providing or selecting these procedures depend on both the user's needs and the properties of such procedures, i.e., the selection process involves analyzing and evaluating their advantages and disadvantages as well as their applicability.

$$A_i = a_i / Max\ a_i \qquad 0<A_i\leq 1, \qquad (1)$$

$$A_i = Min\ a_i/a_i \qquad 0<A_i\leq 1, \qquad (1a)$$

$$A_i = a_i - Min\ a_i/Max\ a_i - Min\ a_i \qquad 0\leq A_i\leq 1, \qquad (2)$$

$$A_i = Max\ a_i - a_i/Max\ a_i - Min\ a_i \qquad 0\leq A_i\leq 1, \qquad (2a)$$

$$A_i = a_i/\sum_{i=1}^{i=n} a_i \qquad 0<A_i<1, \qquad (3)$$

$$A_i = a_i/\sqrt{\sum_{i=1}^{i=n} a_i^2} \qquad 0<A_i<1. \qquad (4)$$

For example, according to [22], procedure (1) is very common and has an intuitive interpretation. It also maintains the proportionality of different values, i.e., $a_i/a_k = A_i/A_k$, for all *i*, *k*. Procedure (2) refines the previous one in order that the normalized scale covers exactly the interval [0, 1], i.e., for each criterion the worse value is 0 and the best value is 1, but in this case the proportionality is not maintained. Procedure (3) offers almost the same advantages as procedure (1), although (3) concentrates A_i towards small values. Finally, procedure (4) offers an important advantage allowing dimensionless comparisons of vectors related to the problem criteria. Procedures (1a, 2a) represent the case of minimum values of procedures (1, 2), respectively, i.e., they vary the relation mentioned above and establish 1 as the worst value and 0 as the best value. An extended comparative analysis of these procedures is out of the scope of the paper, but the reader can refer to [22, 23] for details regarding the different normalization procedures.

4.2 Ranking Algorithms

The ranking service which *WeSSQoS* currently offers includes six ranking algorithms (see equations 5 to 10). As mentioned before, users can also provide their own ranking algorithms. In this sense, users are responsible for selecting the ranking algorithms fulfilling their requirements, by analyzing and assessing the advantages and disadvantages as well as their applicability.

$$Sim_{cosine}(A,B) = \frac{\sum_{i=1}^{n}(a_i * b_i)}{\sqrt{\sum_{i=1}^{n} a_i^2 * \sum_{i=1}^{n} b_i^2}}, \qquad (5)$$

$$Sim_{overlap}(A,B) = \frac{\sum_{i=1}^{n}(a_i * b_i)}{Min\left(\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i\right)}, \qquad (6)$$

$$Sim_{dice}(A,B) = \frac{2*\sum_{i=1}^{n}(a_i * b_i)}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i}, \qquad (7)$$

$$Sim_{jaccard}(A,B) = \frac{\sum_{i=1}^{n}(a_i * b_i)}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i - \sum_{i=1}^{n}(a_i * b_i)}, \qquad (8)$$

$$Dist_{euclidean}(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}, \qquad (9)$$

$$InvDist_{euclidean}(A,B) = \frac{1}{1+\sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}}. \qquad (10)$$

According to [24], the cosine measure (5) assumes that similarity is proportional to the angle between two t-dimensional vectors in a t-dimensional space. Because the numerator is divided by the product of the lengths of the vectors, the measure tends to give low similarities between long vectors, i.e., vectors with many terms. The overlap measure (6) compensates the cosine measure by dividing by the vector having the lowest sum of weights. The Dice coefficient (7) gives more weight to matches in the data than to differences, whereas Jaccard's coefficient (8) is the proportion of characters (i.e., index terms) that match, excluding those characters that lack in both vectors. Finally, the Euclidean distance (9) emphasizes differences between two vectors more than matched features. An important disadvantage of this measure is related to the variables used, i.e., if these variables are correlated then the information provided will be redundant. A variation of the Euclidean distance, emphasizing distance rather than similarity is presented in equation (10). An extended comparative analysis of these algorithms is out of the scope of the paper. Details of such algorithms can be checked in [24, 25].

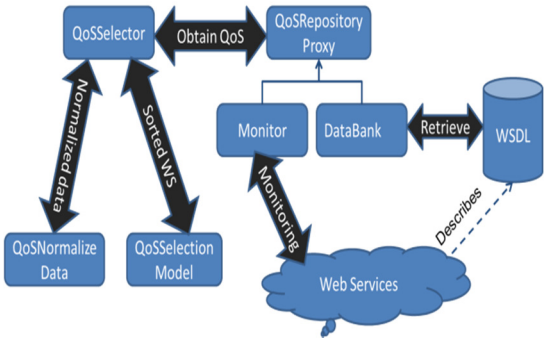

Fig. 2. *WeSSQoS* general architecture

To illustrate the execution of the WS selection process, let's consider the following example. A user needs to select a WS for a given domain with a set of NFRs instrumented in different metrics (e.g., cost, response time, availability, etc.). The user defines the list of values for such metrics in *lreqs* (see Table 2, NFRs from Stakeholders *lreqs*).

In the repository there are 4 WSs that fulfill the functionality required by the user with different QoS (see Table 2, QoS from candidate WSs). Both NFRs and QoS are normalized by applying the procedure that better fulfills the user's needs. In Table 2 we show the results applying the normalization procedure (1). These normalized data are then the input for the *ranking* phase (see Table 3). On top of the table, we depict the results of applying the Euclidean distance algorithm (9). It may be observed that WS1 has the minimum value, thus it looks like a promising candidate for selection before evaluating the compliance degree of the mandatory requirements.

As for the *priority evaluation* phase, let's suppose that all requirements are mandatory. Based on this premise, the results depicted in the bottom of Table 3 shows that WS1 and WS2 comply with 5 of the 8 NFR, whilst WS3 and WS4 comply with 3. When the results obtained by the phases of ranking and priority evaluation are combined, the prioritized list of services is as shown at the bottom of Table 3. WS1 is still the best ranked service, although the ranking results for the rest of services change.

Table 2. Inputs and outputs of the normalization phase

NFRs from Stakeholder, lreqs								
[30, 35, 31, 15, 20, 0.5, 0.03, 150]								
QoS from candidate WSs, WSlist								
WS1	20	30	25	15	10	0.4	0.3	50
WS2	5	10	20	20	15	0.5	0.2	80
WS3	33	11	6	8	10	0.8	0.4	125
WS4	25	35	45	45	15	0.5	0.5	302
Normalized NFRs, lreqsN								
[0.91, 1, 0.69, 0.33, 1, 0.62, 0.60, 0.50]								
Normalized QoS from candidate WSs, WSlistN								
WS1	.7	.9	.6	.3	.5	.5	.6	.2
WS2	.1	.3	.4	.4	.8	.6	.4	.3
WS3	1	.3	.1	.2	.5	1	.8	.4
WS4	.8	1	1	1	.8	.6	1	1

Table 3. Results using ranking and priority evaluation

ID	**Name WS**	**Euclidian distance**	**Ordering by QoS**
WS1	AirportWeather Check	0.71083	1
WS2	BerreWeather	1.14562	4
WS3	FastWeather2	1.11749	3
WS4	Weather	1.01981	2
ID	**Name WS**	**Mandatory**	**QoS vs. Mandat.**
WS1	AirportWeather Check	5/8	1
WS2	BerreWeather	5/8	2
WS3	FastWeather2	3/8	4
WS4	Weather	3/8	3

5 Proposed Framework Architecture

The proposed framework, Web Service Selection Based on Quality of Service (*WeSSQoS*) is structured under the SOA paradigm in order to facilitate its integration into other systems. Figure 2 shows the elements integrating the framework architecture which are described as follows:

- *QoSSelector* is a service that integrates three services: *QoSRepositoryProxy*, *QoS NormalizeData*, and *QoSSelectionModel*, providing a unified view and a single entry point to the whole system.
- *QoSRepositoryProxy* is a service that obtains the QoS of WSs that belong to a given domain. Two sources of QoS information are defined:
  - *Monitor.* It obtains the QoS at execution time by means of monitoring techniques. A monitor works on a predefined catalog of dynamic quality attributes. Any information about static quality attributes will be available in the description of the service, e.g., service cost.
  - *Data Bank*. It obtains the QoS from the WS provider which describes quality data in extended WSDL files. In case of dynamic quality attributes, such as *mean response time*, the quality value is the one that the provider promises to deliver.
- *QoSNormalizeData* is a service that normalizes stakeholder requirements and QoS data obtained from WSs by applying normalization procedures as described in Section 4. Its SOA is flexible enough as to extend the portfolio of normalization procedures. In its current version, *WeSSQoS* provides, but is not limited to, four normalization algorithms. Users can provide and add their own normalization procedures which will be available for the scientific community.
- *QoSSelectionModel* is a service that sorts candidate WSs by applying ranking algorithms as described in Section 4. Also, its internal architecture is flexible enough as to extend the portfolio of ranking algorithms. Currently, *WeSSQoS* provides, but is not limited to, six ranking algorithms. Users can provide and add their own ranking algorithms which will be available for the scientific community.

Figure 2 also shows the relationships among the services previously mentioned. As shown, the composition of services follows an orchestration managed by *QoSSelector* service. A sequence diagram of such orchestration is shown in Figure 3. The main method in *QoSSelector* is *rank4QoSRepository* which is used to rank the services. The input of this operation is a list of repositories (*lProxies*), the list of requirements (*lReqs*), domain of the WSs (*domain*), normalization procedure (*iNumNormalize*), and ranking algorithm (*iNumUtilFunction*). The output obtained is a list of WSs ranked according to the satisfaction of NFRs and mandatory nature, according to the process described in Section 3.

The sequence shown in Figure 3 is described as follows: *rank4QoSRepository* operation invokes *getServicesDataFromDomain* operation for each *QoSRepositoryProxy* (*Databank* or *Monitor*) specified in *lProxies*. From such invocation, the list of services with QoS information is obtained (*WSlist*). In case of having repeated QoS information in more than one repository, a simple priority policy is applied to the repositories list, i.e., the order in the repositories list determines the priority of attributes appearing in more than one repository.

Once the list *WSlist* of services with their QoS information is obtained, the operations of normalization and ranking are applied. First, the operation *getNormalizedData* from *QoS NormalizedData* service is executed. This operation takes as input the following parameters: *WSlist*, NFRs from the stakeholder represented by *lreqs*, and the type of normalization process represented by *iNumNormalize*. The output of this method is the normalized list of QoS and NFR. Afterwards, *QoSSelectionAlgorithm* operation from *QoSSelectionModel* service is executed in order to rank WSs applying the ranking algorithm identified by *iNumUtilFunction*.

The final output is a list of *orderedWS* that can be simple or multiple. A simple list provides WSs sorted by a single ranking algorithm using a single normalization procedure and furthermore provides

Table 4. Interfaces of *WeSSQoS* services

QoSSelector	
Operation: rank4QoSRepository ***Input parameters:*** lProxies: list<Repository Proxy> lReqs: list<Stakeholder Requirements> domain: string iNumNormalize: int iNumUtilFunction: int	***Result:*** orderedWS: list <ServiceData PriorityResult>
QoSRepositoryProxy	
Operation: getServicesDataFromDomain ***Input parameters:*** domain: string	***Result:*** WSList: list <ServiceData>
QoSNormalizeData	
Operation: getNormalizedData ***Input parameters:*** completeWSList: list <ServiceData> lReqs: list <Stakeholder Requirements> iNumNormalize: int	**Result:** NormalizedData: list <normalizedService Data, normalized lReqs>
QoSSelectionModel	
Operation: QoSSelectionAlgorithm ***Input parameters:*** CompleteWSList: list <ServiceData> lReqs: list <Stakeholder Requirements> iNumUtilFunction: int	***Result:*** orderedWS: list <ServiceDataPriorityResult>

WSs sorted by mandatory attributes. On the other hand, a multiple list provides a simple list by each ranking algorithm and normalization procedure applied, considering that stakeholders can provide a list of normalization procedures and ranking algorithms.

Table 4 shows the interfaces of services appearing in the sequence diagram, whereas attributes and classes involved are represented in Figure 4a and 4b. A general description of these elements is provided as follows:

- *lproxies* is a list of repositories from which the *QoSRepositoryProxy* service obtains QoS Data. Each repository has the following information: name, endpoint that corresponds to the URL address where the repository is located (either databank or monitor) and description.
- *lReqs* is a list of NFRs from the stakeholder where each NFR has the following information: name of the quality attribute, required value, and two Boolean values regarding normalization of attributes (maximize or minimize) and mandatory attributes (mandatory or non-mandatory).
- The *domain* is a string that defines a specific class of WSs.
- *Identifiers iNumNormalize* and *iNumUtil Function* represent normalization procedures and ranking algorithms, respectively.

6 *WeSSQoS* Prototype Description

The *WeSSQoS* system described so far is implemented and available in the following URL: http://gessi.lsi.upc.edu/wessqos/. The system has been developed using Java J2EE and Apache Axis2 as web service technology, and Apache Tomcat as the execution platform. We have developed WSs belonging to different domains and placed in different repositories using Glassfish web service technology, in order to assess the technological independence of the platform.

A client Web interface divided into different sections was also developed (see Figure 5). The first section corresponds to *repositories* containing WSs with QoS data description. The basic use case of this section is to provide the domain and repositories over which the search will be done. The domain name is required to obtain a specific subset of services from repositories. The framework allows using both

ISSN 2007-9737

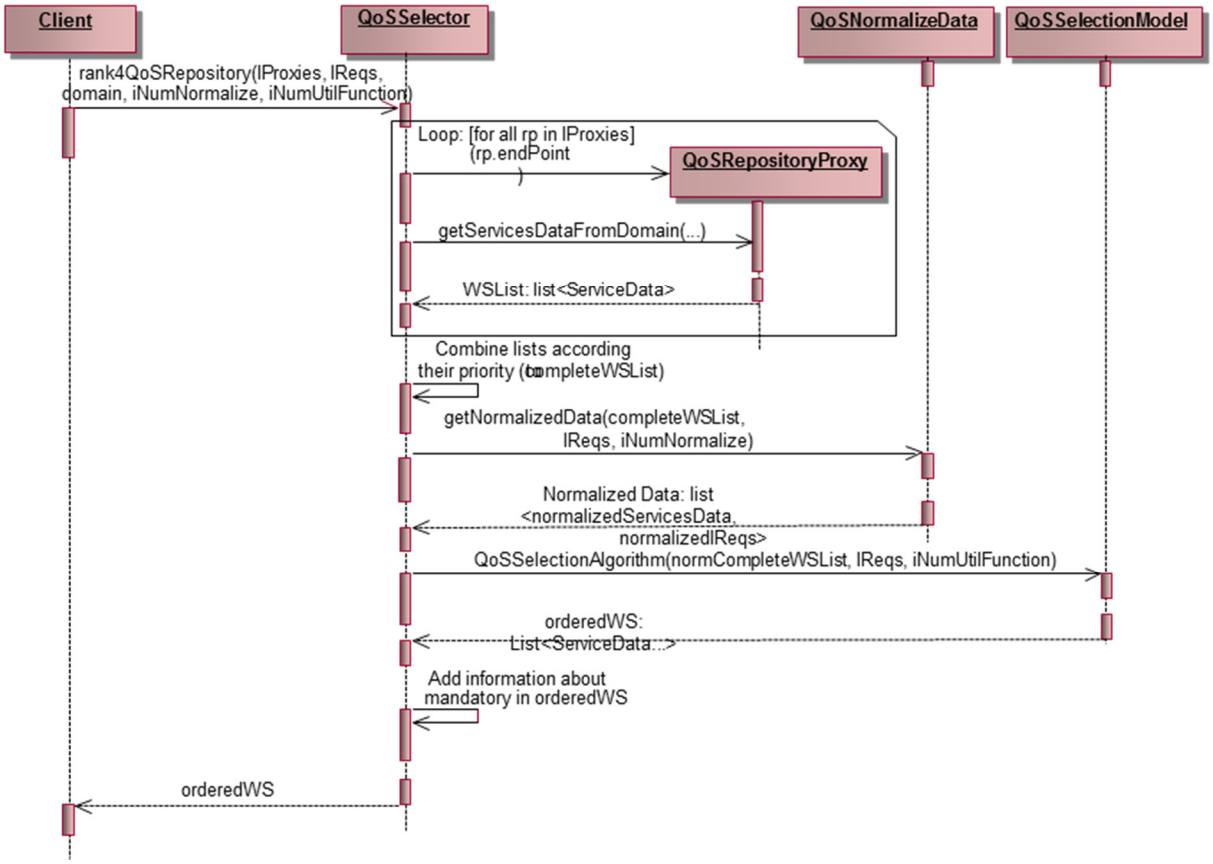

Fig. 3. Sequence diagram of the basic use case of the framework

internal repositories (i.e., local to *WeSSQoS*) and external ones (i.e., provided by stakeholders).

As already mentioned, the repositories are identified using their endpoint. Each repository might have different strategies to extract QoS data, i.e., using the strategy design pattern it is possible to extend the repository behavior adopting different QoS data sources in the same repository (e.g., QoS data from XML documents, databases, etc.). Finally, each repository from the list of chosen repositories can be prioritized.

The second section, depicted in Figure 6, corresponds to *normalization procedures* that will be applied on both QoS data from WS and NFR from stakeholders. The basic use case of this section is to provide at least a normalization procedure in order to compensate the different measurement units of the different QoS and NFR values by projecting them onto a normalized interval. The framework allows using both internal normalization procedures (i.e., local to *WeSSQoS*) and external ones (i.e., provided by the stakeholders).

Normalization procedures are also identified using their endpoint. Each procedure selected or provided might have optional strategies acting as a repository of normalization procedures in the same endpoint.

The third section, depicted in Figure 7, corresponds to *ranking algorithms* that will be applied to prioritize WSs. The basic use case of this section is to provide at least a ranking algorithm fulfilling the data structure specified in the *QoSSelectionModel* service depicted in Table 4.

Furthermore, the framework allows using both internal ranking algorithms and external ones. Ranking algorithms are identified using their

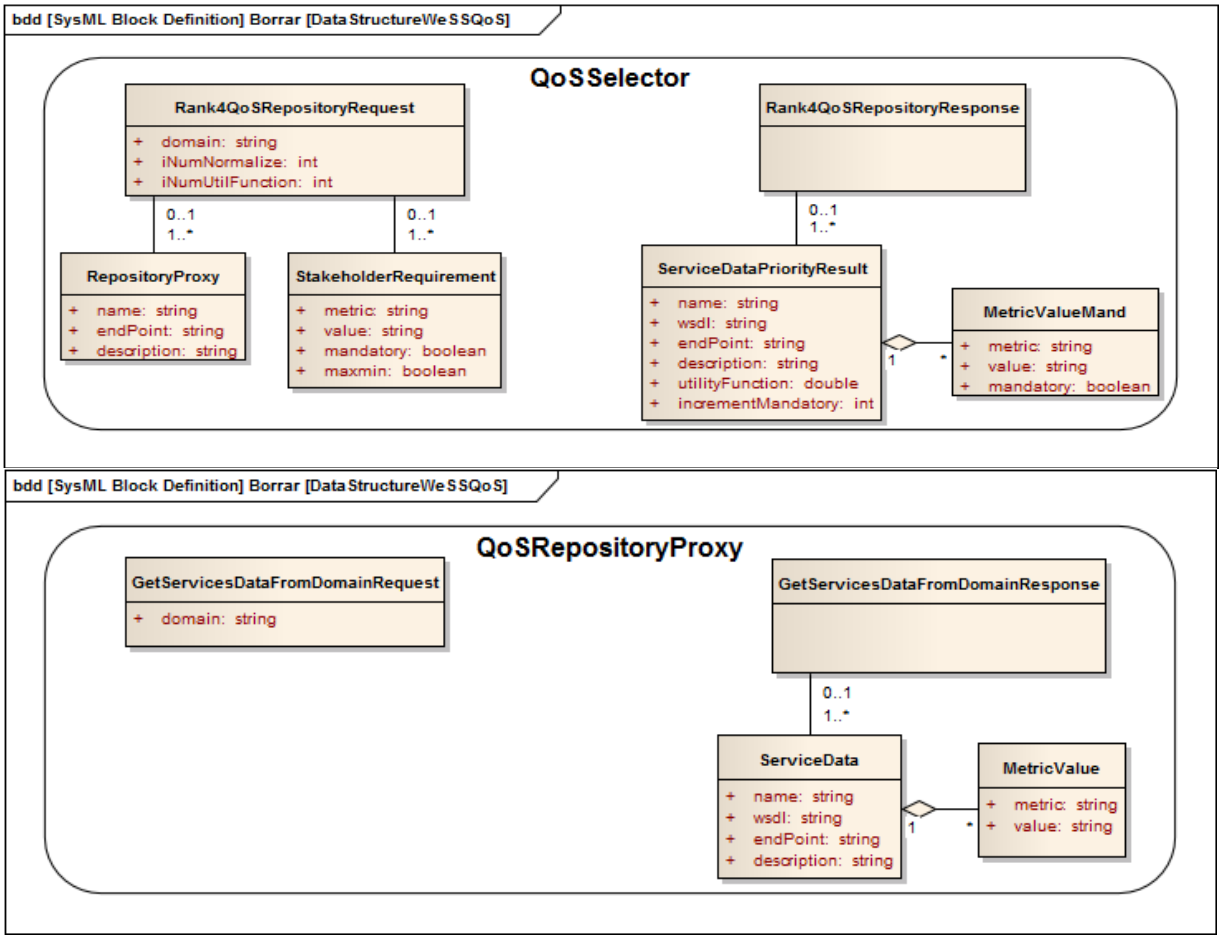

Fig. 4a. Class diagram of the services supporting the internal architecture of *WeSSQoS*

endpoint. Each algorithm selected or provided might have alternative strategies acting as a repository of ranking algorithms in the same endpoint.

Finally, each endpoint from the list of chosen selection models can be deleted.

The fourth section, depicted in Figure 8, corresponds to *stakeholder requirements*, where stakeholders introduce NFRs to be fulfilled. These NFRs are settled over quality attributes that can be attributes provided by the framework based on [5] or by other external source. Clearly, stakeholders have the responsibility of choosing quality attributes that WSs should comply with, i.e., these attributes will be used to compute the relationship (similarity or dissimilarity) between them and the QoS information from WSs. For each attribute introduced, the following information is required: the value that WSs should meet, maximization or minimization to compensate the attribute value, and information that allows identifying when an attribute is mandatory to prioritize services.

Finally, the results section depicted in Figure 9 shows the resulting ranking and provide different options described below. The first ranking provided is a sorted WSs list according to the ranking algorithm and normalization procedure chosen by stakeholders. Also the number of mandatory attributes fulfilled is depicted.

Figure 10 shows the *graphic* option of the results with two types of charts. The chart on the

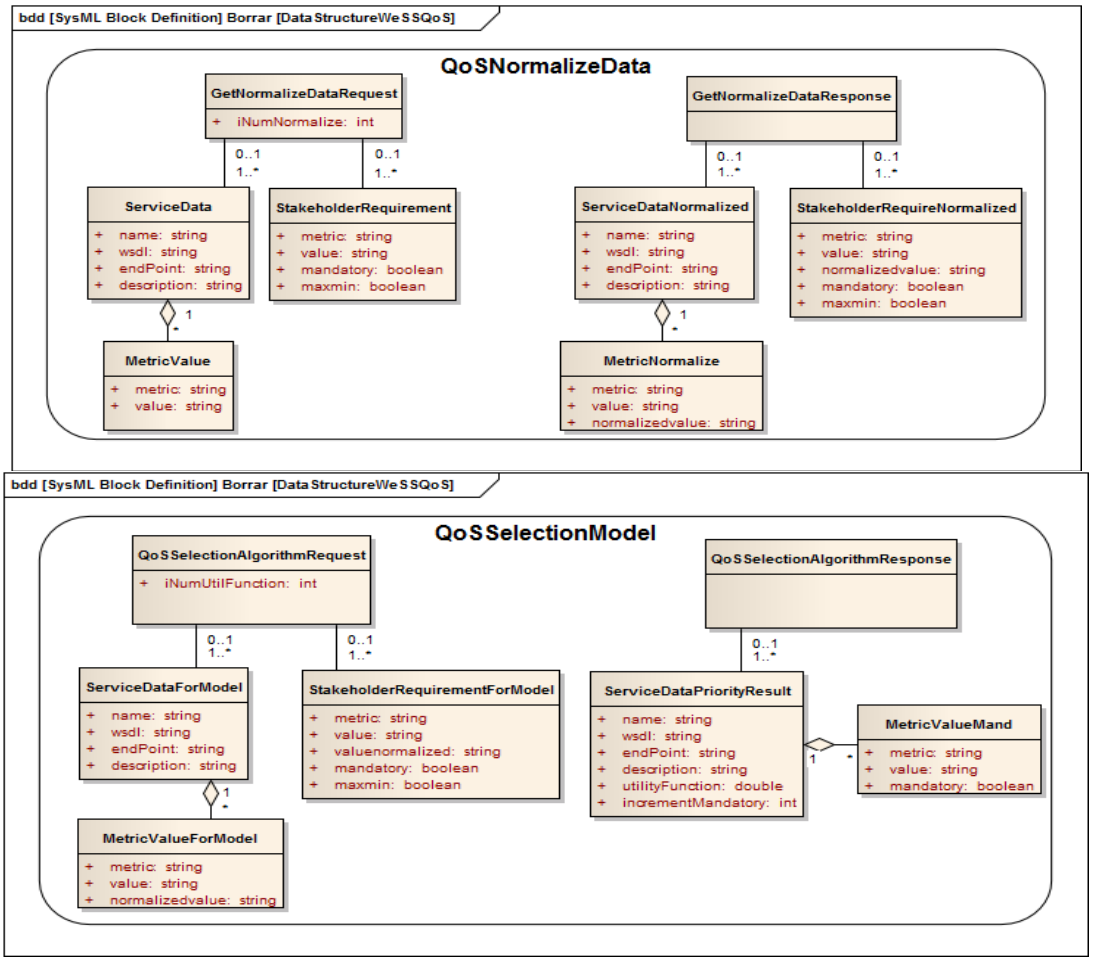

Fig. 4b. (cont.) Class diagram of the services supporting the internal architecture of *WeSSQoS*

right shows the ranking results applying normalization procedure (1) and ranking algorithm (5). The chart on the left shows the ranking results applying mandatory requirements.

As mentioned before, the architecture of *WeSSQoS* allows providing both a list of normalization procedures and a list of ranking algorithms supplying the list of results. This functionality allows comparing the different rankings obtained and the behavior shown by a particular ranking algorithm in combination with a normalization procedure. In this sense, Figure 11 shows the ranking of four services applying two ranking algorithms with two normalization procedures yielding four different results.

7 Validation

In order to test our prototype, we designed a scenario to execute some test cases. The scenario was designed to assess the following features of our framework:

- *Quality attributes management*. In the scenario, the customer can decide the quality attributes which she is interested in. These attributes may or may not be defined in the information about the WSs being selected. The basic case is when the customer asks for a subset of attributes defined on the repositories. The customer can also ask for

Fig. 5. Repositories of web services with QoS description

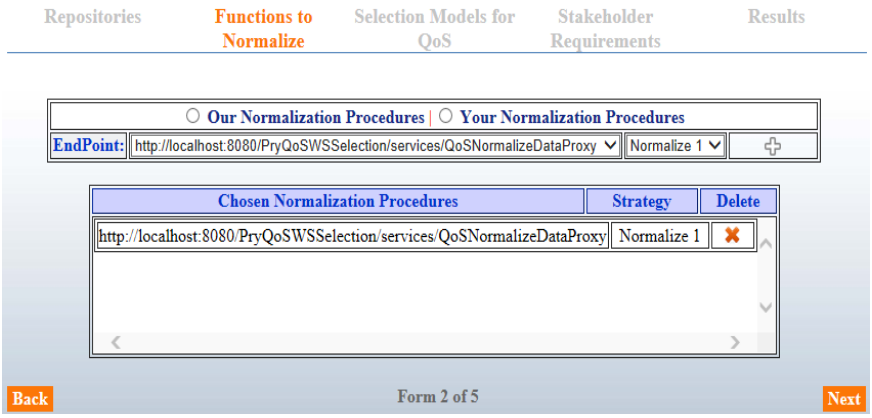

Fig. 6. Normalization procedures interface

attributes that are not specified on repositories, these attributes will be treated as undefined by the ranking algorithm.

- *Repositories independence.* Our framework does not have restriction on the number of repositories used for the search. Each repository can be static or dynamic. When there is more than one repository, the following assumptions are considered:
  - The WS of each repository can be different. In this case we consider as WS candidates the union of all services inside all repositories.
  - More than one repository may contain information of a given WS, but the quality attributes are disjoint. In this case, the algorithm will simply combine the required attributes retrieving them from the adequate repositories.

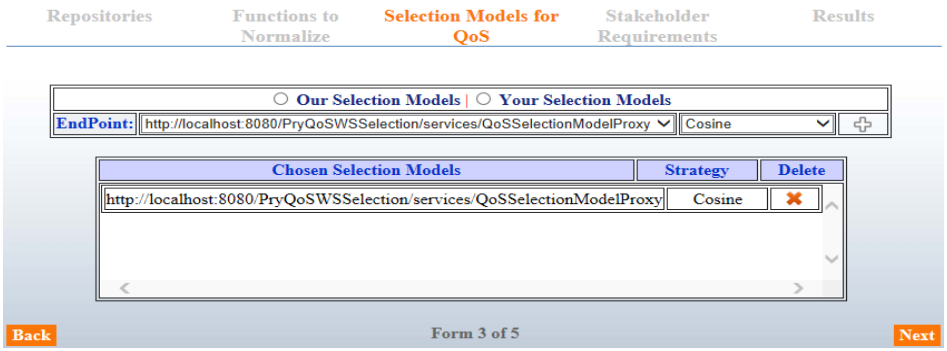

Fig. 7. Ranking algorithms interface

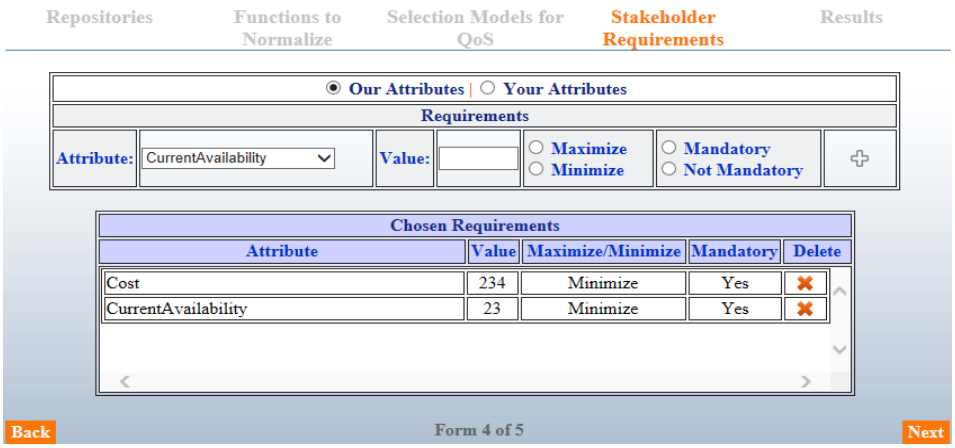

Fig. 8. Stakeholder requirements interface

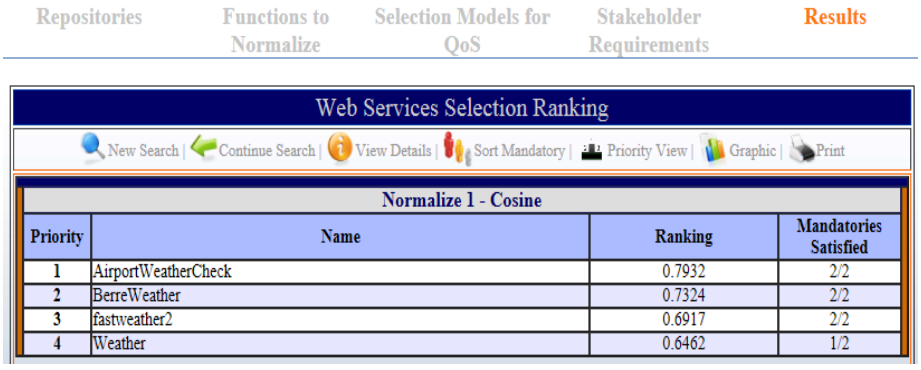

Fig. 9. Interface for representation of results

- More than one repository may contain information of a given WS, and some quality attribute may appear in more than one repository. In this situation, the value is taken from the repository with a higher priority (i.e., the one declared first).

Figure 12 shows the architecture implemented and the necessary data for running the tests previously described. We have both types of QoSRepositoryProxy (static and dynamic). The Monitor instances use Axis, whilst the DataBank (which contains information about two WS

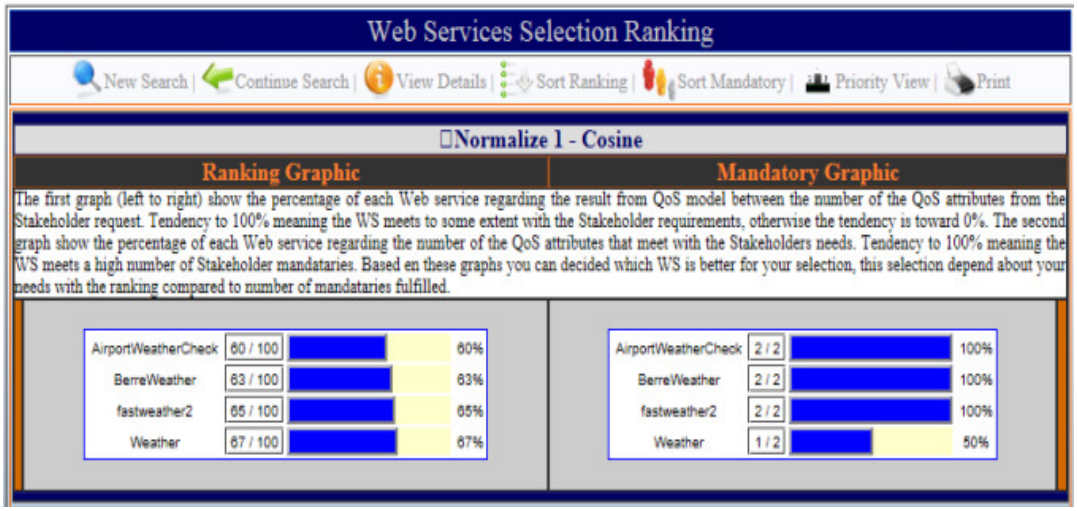

Fig. 10. Ranking results using Euclidean distance

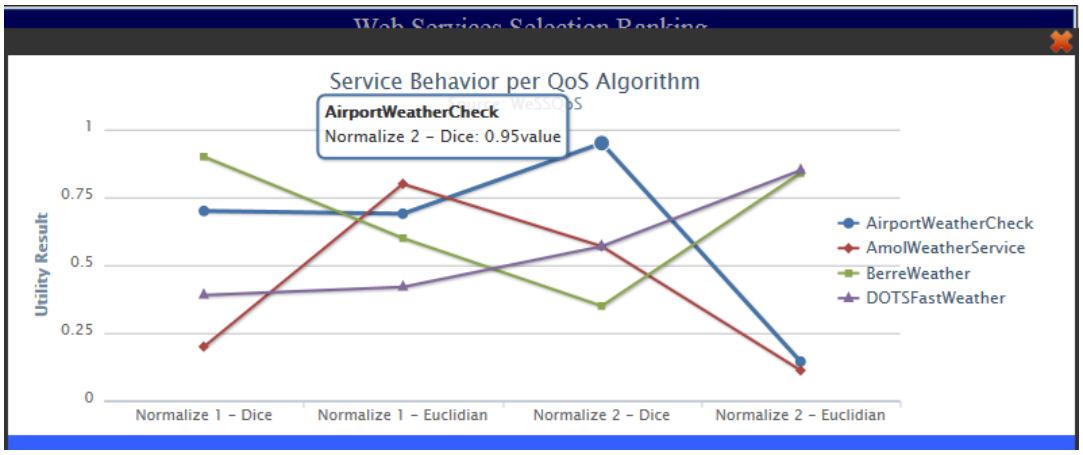

Fig. 11. Ranking results considering mandatory requirements

domains) uses Glassfish. In Figure 12, the names of some of WSs were included. These services were selected in order to highlight services located in more than one repository, and some of them have attributes in more than one repository.

Databank1 contains information about all attributes with the exception of Current ResponseTime (CRT) and CurrentAvailability (CA). In the services from Monitor1 and Monitor2, the information about what attributes have information is included too. In addition to the CRT and CA, there is also information about the AverageResponseTime (ART) in some services.

If the priority of repositories (i.e., their order of appearance) is Monitor1, Monitor2, DataBank1, given the service AirportWeatherCheck (which is located in all the repositories), ART, CRT, and CA will be taken from the Monitor1, and the other attributes, from the DataBank1.

However, if the order was Monitor2, Monitor1, and DataBank1, the CRT would be taken from the Monitor2, ART and CA, from the Monitor1, and the rest, from the DataBank1. The users can test the scenarios described before or test other ones using the WeSSQoS prototype.

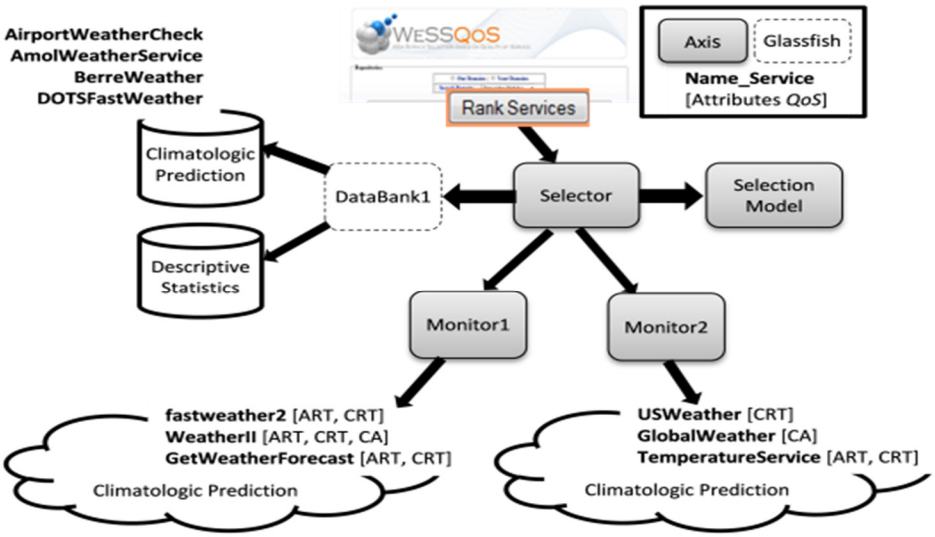

Fig. 12. Scenario for *WeSSQoS* tests

8 Conclusions and Future Work

In this paper we presented *WeSSQoS*, a framework for ranking available WSs through the evaluation of their QoS with respect to the stated NFRs. In terms of the criteria introduced in Section 2, we can conclude that the proposal has the following advantages:

- *Architectural style. WeSSQoS* is developed as a Service Oriented System itself. Following SOA principles, users can add new services related to ranking algorithms, repositories and normalization procedures, if they are compliant with the expected service definitions.
- *Quality attributes. WeSSQoS* is independent of the Quality Model or ontology used to define quality attributes. The system interface allows users to select from a well-known predefined set of attributes based on [5], and also add any kind of quality attributes from any quality model. As many frameworks, *WeSSQoS* is able to work with either static or dynamic quality attributes, although it's important to mention that this distinction is implicit in the way the data are retrieved.
- *QoS data. WeSSQoS* is able to retrieve quality attributes from either quality descriptions in service definition (WSDL) or by monitoring systems. The usage of a common interface (proxy) to retrieve data in a uniform way from these sources provides extensibility to add new kinds of repositories, independently of the approach used to obtain the data.
- *Multinormprocedure. WeSSQoS* is able to work with any kind of normalization procedure that is implemented using the defined interface. Eventually, we could use arbitrarily complex procedures, e.g., aggregators of results through choreography of other WSs defining different normalization procedures.
- *Multialgorithm. WeSSQoS* is able to work with any kind of ranking algorithm that is implemented using the defined interface. Eventually, we could use arbitrarily complex algorithm, e.g., aggregators of results through choreography of other WSs that define different algorithms.
- *Multirepository. WeSSQoS* allows the user to include several repositories of WSs with independence of the technology used. Furthermore, it provides a mechanism to combine the QoS data when the same service is present in more than one repository. Currently, the user is responsible for selecting those repositories that are compatible with each other, e.g., repositories should use a

common terminology to refer to the same quality attribute.
- *Prototype available. WeSSQoS* is available at http://gessi.lsi.upc.edu/wessqos/. The current version has been tested and validated as explained in Section 7.

In Section 5 we dealt with the issue concerning WS repositories' priority policy, the main idea of this is to integrate in a general repository the WSs coming from all chosen repositories in a prioritized way. It is worth noting that WS integration is used in repositories combination and it is not part of WS composition, this topic is out of the scope of the paper.

As future work, we identified several research lines and improvements that could be performed in order to increase the current framework's capabilities:

- Perform tests in large web service ecosystems to ensure the correctness and suitability of the framework to rank web services in real situations.
- Increase the number of dynamic quality attributes retrieved by the monitoring system.
- Design different sophisticated mechanisms to combine data from several repositories and unify these strategies under a common interface in order to build it as a service.
- Automate analysis and evaluation of ranking algorithms and normalization procedures.

Acknowledgements

This work was partially supported by the Spanish project TIN2013-44641-P. Oscar Cabrera Bejar is a Ph.D. student at the UPC using a CONACYT grant.

References

1. **Taylor, S., Iqbal, M., & Nieves, M. (2011).** *ITIL Version 3 Service Strategy.* The Office of Government Commerce.
2. **Papazoglou, M. (2007).** *Web Services: Principles and Technology.* Pearson-Prentice Hall.
3. **Menasce, D. (2002).** QoS issues in web services. *Internet Computing, IEEE,* Vol. 6, No. 6, pp. 72–75.
4. **Robertson, S. & Robertson, J. (2012).** *Mastering the Requirements Process: Getting Requirements Right.* Pearson Education.
5. **Ameller, D. & Franch, X. (2008).** Service level agreement monitor (SALMon). *Seventh International Conference on Composition-Based Software Systems, ICCBSS'08,* IEEE, pp. 224–227.
6. **Al-Masri, E. & Mahmoud, Q.H. (2009).** A broker for universal access to web services. *Seventh Annual Research Conference on Communication Networks and Services, CNSR'09,* IEEE, pp. 118–125.
7. **Yu, T. & Lin, K.J. (2005).** Service selection algorithms for Web services with end-to-end QoS constraints. *Information Systems and E-Business Management,* Vol. 3, No. 2, pp. 103–126.
8. **Yu, T. & Lin, K.J. (2005).** A broker-based framework for QoS-aware web service composition. *IEEE International Conference on E-Technology, e-Commerce and e-Service, EEE'05,* IEEE, pp. 22–29.
9. **Wang, X., Vitvar, T., Kerrigan, M., & Toma, I. (2006).** A QoS-aware selection model for semantic web services. *Service-Oriented Computing (ICSOC 2006),* Springer, pp. 390–401.
10. **D'Mello, D.A., Ananthanarayana, V.S., & Santhi, T. (2008).** A QoS broker based architecture for dynamic web service selection. *Second Asia International Conference on Modeling & Simulation, AICMS'08,* IEEE, pp. 101–106.
11. **Wang, H.C., Lee, C.S., & Ho, T.H. (2007).** Combining subjective and objective QoS factors for personalized web service selection. *Expert Systems with Applications*, Vol. 32, No. 2, pp. 571–584.
12. **Wang, P., Chao, K.M., & Lo, C.C. (2010).** On optimal decision for QoS-aware composite service selection. *Expert Systems with Applications*, Vol. 37, No. 1, pp. 440–449.
13. **Mohanty, R., Ravi, V., & Patra, M.R. (2010).** Web-services classification using intelligent techniques. *Expert Systems with Applications*, Vol. 37, No. 7, pp. 5484–5490.
14. **Tao, Q., Chang, H.Y., Gu, C.Q., & Yi, Y. (2012).** A novel prediction approach for trustworthy QoS of web services. *Expert Systems with Applications*, Vol. 39, No. 3, pp. 3676–3681.
15. **Cai, H., Hu, X., Lü, Q., & Cao, Q. (2009).** A novel intelligent service selection algorithm and application for ubiquitous web services environment. *Expert Systems with Applications*, Vol. 36, No. 2, pp. 2200–2212.
16. **Sha, L., Shaozhong, G., Xin, C., & Mingjing, L. (2009).** A QoS based web service selection model. *International Forum on Information Technology*

and Applications 2009, IFITA'09, IEEE, pp. 353–356.

17. **Alrifai, M., Risse, T., Dolog, P., & Nejdl, W. (2009).** A scalable approach for QoS-based web service selection. *Service-Oriented Computing (ICSOC'08) Workshops,* Springer, pp. 190–199.
18. **Huang, A.F., Lan, C.W., & Yang, S.J. (2009).** An optimal QoS-based Web service selection scheme. *Information Sciences*, Vol. 179, No. 19, pp. 3309–3322.
19. **Lin, C.F., Sheu, R.K., Chang, Y.S., & Yuan, S.M. (2011).** A relaxable service selection algorithm for QoS-based web service composition. *Information and Software Technology*, Vol. 53, No. 12, pp. 1370–1381.
20. **Gao, Z.P., Chen, J., Qiu, X.S., & Meng, L.M. (2009).** QoE/QoS driven simulated annealing-based genetic algorithm for Web services selection. *The Journal of China Universities of Posts and Telecommunications*, Vol. 16, pp. 102–107.
21. **Salton, G., Wong, A., & Yang, C.S. (1975).** A vector space model for automatic indexing. *Communications of the ACM,* Vol. 18, No. 11, pp. 613–620.
22. **Barba Romero, S. & Pomerol, J. (2000).** *Multicriterion Decision in Management. Principles and Practice*. Kluwer Academic Publishers.
23. **Peña, V.H., Lai, T.L., & Shao, Q.M. (2008).** *Self-normalized processes: Limit theory and Statistical Applications*. Springer.
24. **Knappe, R. (2005).** *Measures of semantic similarity and relatedness for use in ontology-based information retrieval*. Doctoral dissertation, Roskilde University.
25. **Bernstein, A., Kaufmann, E., Kiefer, C., & Bürki, C. (2005).** *Simpack: A generic java library for similarity measures in ontologies*. University of Zurich.

Oscar Cabrera is a Ph.D. student in Computer Science at the Universitat Politècnica de Catalunya, UPC, Barcelona, Spain. He obtained his M.Sc. degree in Computing in the Software Engineering area from the National Center for Research and Technological Development (CENIDET), Cuernavaca, Morelos, Mexico. He is a member of the GESSI research group at UPC. His current research lines include service-oriented computing, current trends in smart cities, context modeling, quality models, and information technology in software development.

Marc Oriol is a Ph.D. student in Computer Science at the Universitat Politècnica de Catalunya, UPC, Barcelona, Spain. He obtained his M.Sc. degree in Computing from the same university. He is a member of the GESSI research group at UPC. His current research lines include service-oriented computing, quality-of-service, and monitoring.

Xavier Franch is Associate Professor and Head of the GESSI research group at the Universitat Politècnica de Catalunya, UPC, Barcelona, Spain. He obtained his Ph.D. and M.Sc. in Informatics from this University. His current research lines include service-oriented computing, requirements engineering, software quality, and software architecture, among others.

Jordi Marco is Associate Professor and member of the GESSI research group at the Universitat Politècnica de Catalunya, UPC, Barcelona, Spain. He obtained his Ph.D. and M.Sc. in Computing from this University. His current research lines include service-oriented computing, conceptual modeling, container libraries, and computer graphics.

Lidia López is a researcher of the GESSI research group at the Universitat Politècnica de Catalunya, UPC, Barcelona, Spain. She obtained her Ph.D. and Engineer degree in Informatics from this University. Her current research lines include service-oriented computing, goal-oriented modeling, and open source software.

Olivia Graciela Fragoso Díaz is a researcher in the Software Engineering area at the National Center for Research and Technological Development (CENIDET), Cuernavaca, Morelos, México. Her areas of interest are web services selection, web services for e-learning, software reusability, and processes for software development.

René Santaolaya Salgado is a researcher in the Software Engineering area at the National Center for Research and technological development (CENIDET), Cuernavaca, Morelos, México. His areas of interest are web services, software reusability, and integrated environments for software development visual programming.

Article received on 10/08/2014; accepted on 01/11/2014.