PrefWS3: Web Services Selection System Based on Semantics and User Preferences

Rohallah Benaboud
Department of Mathematics and Computer Science, University of Oum El Bouaghi, Algeria
LIRE Laboratory, University of Constantine 2 - Abdelhamid Mehri, Constantine, Algeria
E-mail: r_benaboud@yahoo.fr

Ramdane Maamri and Zaidi Sahnoun
LIRE Laboratory, University of Constantine 2 - Abdelhamid Mehri, Constantine, Algeria
E-mail: rmaamri@yahoo.fr, sahnounz@yahoo.fr

Keywords: semantic web service discovery, domain ontology, OWL-S, QoS, user preferences, reputation

Received: January 21, 2016

With the growing number of web services on the Web, many approaches have been proposed to help users discover and select the desired services. Nevertheless, finding the best service that meets the user needs and preferences is still a problem. In this article, we introduce a user preferences based semantic web services discovery and selection system (PrefWS3). PrefWS3 is considered to be a user-centric system which helps users in formulating their requirements and preferences. This system involves semantic enhancement of both request and web services and provides an efficient semantic-based matching mechanism, which calculates the semantic similarity between the request and the web service. Furthermore, PrefWS3 includes a QoS-aware process and provides a reputation mechanism that enables users to evaluate the credibility of the web services they use. In this article, we also present the results of a comparison of the PrefWS3 and some other published approaches to evaluate its effectiveness.

Povzetek: Prispevek obravnava izbiro spletnih storitev na osnovi semantike in preferenc uporabnika.

1 Introduction

Web services have emerged as a key technology for implementing Service Oriented Architectures (SOA), aiming at providing interoperability among heterogeneous systems and integrating inter-organization applications [1]. Web services are designed to be selected via discovery mechanisms. Web Service discovery mechanisms include a series of registries, indexes, catalogues, agent-based and Peer to Peer solutions. The most dominating among them is the Universal Description Discovery and Integration (UDDI) [2] which is essentially based on keywords search on WSDL descriptions of Web services. Simple keyword matching does not capture the underlying semantics of web services [3]. As a result, only the services which have same syntactic description with the user request may be considered for selection. For example, when searching services with the keyword 'vehicle', the ones whose descriptions contain the word 'car' will not be returned. Thus, the discovery process is also constrained by its dependency up on human intervention in choosing the appropriate service based on its semantics [4].

In order to solve the above-mentioned problem, a variety of conceptual models have been proposed over these past years to add semantics to Web Services descriptions. These include WSDL-S [5], WSMO [6], and OWL-S [7]. These so-called Semantic Web services (SWS) are Web services that are annotated with semantic descriptions. This semantic is made through ontologies; one of the important technologies of the Semantic Web.

The discovery of SWS is mainly based on their functional aspects (Inputs, Outputs, Pre-conditions and effects). However, due to the increasing availability of Web services that offer similar functionalities, other parameters have to be considered during the discovery process, especially user preferences that are expressed in term of constraints on quality of service (QoS), i.e., execution time, cost, reliability, availability, etc.

Several approaches of Web Services discovery have been proposed in the literature; however, finding the best and the right web service that meets user needs and preferences is still a problem. This is due to a number of challenges. Some of them include [4] [8]:

- Descriptions of the vast majority of already existing web services are specified using WSDL and do not have associated semantics.
- From the user's point of view, expressing a request can be a disturbing burden, because he may not have the required expertise or skills.
- Searching is a simple keyword based search; as a consequence, matching does not capture the underlying semantics of web services.
- Accurate service matchmaking for service discovery can be computationally very expensive.
- Dishonest service provider may advertise fake QoS.

In this paper, we present a complete system for web service discovery and selection named PrefWS3, which is able to cope with most of the challenges mentioned

above. The proposed system covers the entire spectrum of tasks from service request to service invocation, and also after service invocation.

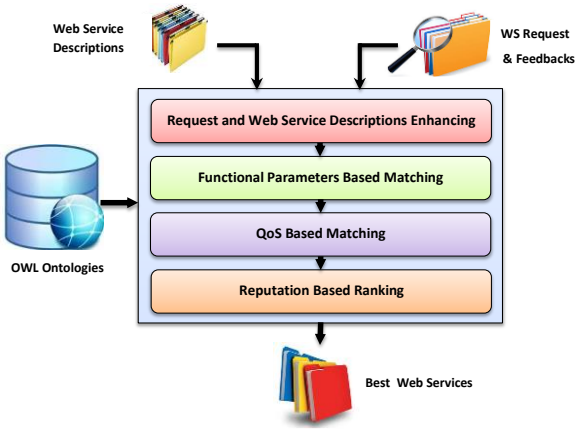

Figure 1: The key steps of PrefWS3.

Figure 1 illustrates the key steps of PrefWS3: The first step involves semantic and QoS enhancing of the request and web service description. The second step deals with the functional parameters based matching of the request against the advertisement services. In the third step, we perform a QoS based matching. In the last step, the user feedback is taken into account for the selection of the best web services. These steps make PrefWS3 a cascading filtering mechanism that finds the best web services from a set of raw web services.

Several approaches have discussed separately the previous four steps, but not all at the same approach. In addition, these approaches differ in the way of each step is implemented. PrefWS3 aims to provide a more "user-centric" system simplifying the service discovery using semantics while satisfying QoS requirements, and to free users from time consuming human computer interactions and Web search. To show the effectiveness of PrefWS3, we compare it with other approaches. The contributions of this paper regarding the different steps can be summarized as follows:

1) Request and web service descriptions enhancing:
- Enhancement of OWL-S profile with QoS information.
- Provide users a way to specify their requirements and preferences expressively and flexibly.

2) Functional parameters based matching: Presenting an efficient matchmaking mechanism that captures semantic similarity between requests and services in a more efficient way with less time.

3) QoS-aware service selection:
- Provide a QoS based filtering mechanism that aims to filter out services that do not meet the service user preferences.
- Introduce a QoS monitoring mechanism that aims to measure the QoS values in order to verify whether the measured values comply with QoS values published by the Web service provider.

4) Reputation based ranking: Provide a mechanism that gives confidence to a user when selecting a web service.

In a previous paper [9], we have addressed some aspects of the PrefWS3 system. The present paper extends the last one by introducing new important mechanisms such as WSLD2OWLS translating, QoS weights calculating, and QoS monitoring mechanisms. Furthermore, the ontology-based OWL-S extension, the QoS based services filtering and the reputation mechanisms, which are previously addressed, are extended in order to make the service selection process more accurate and practical.

The rest of the paper is organized as follows: We present the related works in Section 2. Section 3 gives an overview of the proposed system, and section 4 provides a detailed discussion on request and web services descriptions enhancing. The detailed description of functional parameters based matching is presented in Section 5. Section 6 and 7 include a discussion on QoS based matching and Reputation based ranking respectively. The evaluation of the proposed system is presented in Section 8. Finally, a conclusion and future work are presented in Section 9.

2 Related works

Researches in Web services discovery have been necessary since the number of available services on internet has increased and the user gets tired to find desired service. In this section, we present and analyze the related works in order to comprehend the benefits that may be obtained and to put our contributions in the context of service web discovery.

Most current approaches for web service discovery depend on the measurement of the similarity degrees between service request and service advertisement. The work in [10] presents a matchmaking algorithm which compares input and output concepts of the user request to the service description and defines four levels of matching: Exact, Plug in, Subsumes, and Fail. However, the use of such discrete scale classification of matching is not sufficient to best rank services. Some of the relevant services might be eliminated due to not fitting those discrete scales. PrefWS3 calculates the total similarity score of the web services according to their relevancy to the user request. OWL-S matchmakers are the mainstream in contemporary SWS matchmakers [11]. iSeM [12] performs structural matching between the signatures of a given Web service and request using the logic-based input/output concept matching, the text similarity-based approach, the ontology-structure-based approach, and the SVM-based approach and, after that, adjusts its aggregation and ranking parameters using machine learning. iMatcher2 [13] combines the SPARQL query language for logical referencing and the syntactic similarity measure to calculates the degree of semantic matching between two OWL-S service profiles. OWLS-MX3 [14] takes into account the shortest distance and the common parent classes between the concepts in an

ontology to compute the semantic similarity between input/output concepts of service and requests.

The work in [15] introduces a Semantic Advanced Matchmaker (SAM), which provides ranking and scoring based on concept similarity. The authors created their own similarity distance ontologies to find the distance between objects. This ontology is supposed to contain proper similarity scores through the assignment of concept-similarity ratings of all the concepts in the ontology by a similarity ranking mechanism. They perform the matchmaking considering the input/output interface of services. In [16], the authors present a semantic matching approach for discovering semantic web services through a broker-based semantic agent (BSA). The BSA performs semantic matching according to the concepts meanings, the concepts similarities, and distance of concept relations. The semantic distance calculation is based on subsumption-based similarity and hasSynonym, hasIsa relationships. Against the two latter works, PrefWS3 don't use only the subsumption relationships between concepts to calculate their similarity but it also takes into account common properties between them. Additionally, the semantic distance between ontology concepts is not necessarily determined according to the distance between concepts. Two concepts that are directly attached may be semantically very different. This case may take place when a concept extends another one by introducing several new properties. In the paper [17], the authors present the application of Case-based Reasoning (CBR) to the problem of service discovery and selection by introducing a case representation, learning heuristics and different similarity functions. The proposed approach combines notions of CBR with the use of WordNet as lightweight semantic basis. The major disadvantage of CBR is that users might rely on previous experience without validating it in the new situation [18]. This is clearly a problem in changing web services functionalities where past descriptions may not reflect current descriptions. In addition, the system requires a large memory space to store all the previous cases in the form of problem-solution pairs. Semantic web service technology is already adopted in several web based applications and solutions, the authors of [19] propose an intelligent system in order to facilitate semantic discovery and interoperability of Web Educational Services that manage and deliver Web media content.

Unlike the aforementioned matchmakers, the matchmaking mechanism of PrefWS3 takes into account the role of concepts in the request and the web service, i.e, concepts are inputs or outputs, to calculate the degree of similarity between them. We think that an output in the request should not be considered as similar to a more generic output in the advertised service, and an input in the advertised service should not be considered as similar to a more generic input in the request.

Since there are many functionally similar Web Services available in the Web, it is an absolute requirement to distinguish them using a set of non-functional criteria such as Quality of Service (QoS). The work in [20] presents a QoS-based model for web service discovery by extending the UDDI's data structure types in order to enhance UDDI model with QoS information. However service discovery and selection are still done by human consumer. Furthermore, this approach is impractical with a huge number of Web services available for selection. The authors of [21] propose a QoS-based web service selection system that handles QoS requests with both exact values and fuzzy values, return two categories of matching offers: super-exact and partial matches. In [22], users' preferences are defined by a lexical ordering in accordance with their perceived importance. They presented an algorithm to compare web services based on their qualities (QoS). According to the proposed algorithm, a web service WS1 is considered better than a web service WS2 if the first QoS attribute that distinguishes between WS1 and WS2 ranks WS1 higher than WS2. This algorithm is simple, but if the user indicates that tow preferences are equally important, the algorithm will take into account only the first preference in the lexical ordering and ignores the second preference. In our system, the use of the weighted sum method allows to take into account all preferences each with its importance degrees. The authors of [23] present a web service selection framework, which takes into account implicit preferences that are inferred from information related to context and profile of the user. Similarities between different attributes of service request and service are captured thanks to fuzzy-set-based techniques. It is argued that augmenting the user's query by preferences depending on their context and their profile allows for highly improving the result's quality. However, the proposed framework requires too much information and a large number of fuzzy rules to infer implicit preferences in each specific domain. Moreover, the proposed framework doesn't take into account the importance of each QoS.

A major problem in using QoS for service discovery is the specification and storage of the QoS information. Most of QoS-aware discovery mechanisms, described above, ignore this problem. In our work, we propose an ontology-based OWL-S extension to add QoS to OWL-S descriptions. Furthermore, PrefWS3 proposes a QoS-based filtering mechanism which filters out services that do not meet the user preferences described as QoS constraints. This filtering mechanism takes into account the degree of confidence that the user has on the specified constraints.

Some approaches already exist about involving the user in the process of service discovery. Those approaches are variants of reputation systems in which the users rate the service providers and share these ratings with other users. The work in [20] presents a reputation-enhanced model that contains a reputation manager which assigns reputation scores to the services based on user feedback regarding their performance. Then, a discovery agent uses the reputation scores for service matching, ranking and selection. The authors of [24] introduce a Web service selection mechanism based on user ratings and collaborative filtering. Services are ranked based on similarity to the user's ratings from the collected feedback database from the users. The

similarity mechanism is calculated based on Pearson correlation coefficient. A Bayesian network trust and reputation model for web services is introduced in [25], which considers several factors when assessing web services' trust: direct opinion from the truster, user rating (subjective view) and QoS monitoring information (objective view). In [26], the authors present a QoS-based semantic web service selection and ranking solution with the application of a trust and reputation management method, which detects and deals with false ratings by dishonest providers and users.

In most of the aforementioned reputation mechanisms, the satisfaction criterion of the rater is unknown since the service user gives one rating score for all QoS of the invoked service. Without knowing the intention of the rater, it is almost impossible to make a given rating meaningful. In the reputation mechanism of PrefWS3, the service user gives a rate score for each QoS attribute of the used Web services. Furthermore, PrefWS3 gives more importance to recent ratings.

3 PrefWS3 overview

PrefWS3 aims to provide a complete system for web service selection that simplifies the service discovery using semantics while satisfying the user preferences. PrefWS3 covers the entire discovery process through several components that take charge of the request and the web services descriptions process, functional based matchmaking, filtering and matching of quality of service parameters, and reputation and rating mechanism. There are four types of data needs to be collected in PrefWS3 to deal with the service request: Web service description in both WSDL or OWL-S files, public open OWL ontologies on the Internet, QoS data, and ratings.

Figure 2 illustrates the main components of the PrefWS3 system, which are described as follows:

Interface Management: This module manages the different interactions with the system users. Depending on the nature of the user and type of his request, the required scenario occurs. These requests include the following parts:

- Requests from service providers in order to register their web services.
- Requests from service consumers, which can be either web services lookup requests or service ratings.

WSDL to OWL-S translating: A large number of service providers use WSDL based syntactic description to describe their services. Therefore in our system, we use a translator to translate WSDL files into OWL-S files and provide semantically enriched description. The translating mechanism uses domain ontologies for mapping complex types, inputs, and outputs of WSDL into OWL ontology concepts.

OWL-S Extending with QoS Information: We use OWL-S service profile as a model for semantic annotation of Web service descriptions. However, OWL-S mainly focuses on describing functional aspects of a Web service and does not describe QoS aspects. After the translation from WSDL files to OWL-S files, the OWL-S profile must be enhanced with QoS information to enable selecting the best services that meet user preferences.

OWL-S repository: This component is responsible for storing the semantic service descriptions as OWL-S files, which could be used for service discovery process.

QoS Weight Calculation: Service consumers have different preferences. For example, a service consumer may want a Web service with lower cost while for another one; the execution time could be his most important parameter. For this raison, we propose that the service consumer may specify that a QoS attribute is more important than another one. Indeed, a weight is given for each QoS attribute. Weights are in [0, 1] where higher weights represent greater importance. Because weights are an important factor for determining the overall quality of a Web service, we calculate the weights for each QoS attribute according to the service consumer QoS preferences. However, distributing the weight of many QoS attributes overburdens the service consumer. Weights calculation can be consider as a multiple decision criteria problem and therefore, we can apply an Analytic Hierarchy Process (AHP) which becomes one of the best known and most widely used multi-criteria decision making methods [27]. By applying this method, the system can easily calculate the weights because it requires only a simple evaluation between two QoS attributes.

Request Generating: In PrefWS3, a request is described in the same manner as a service to facilitate the matching of their descriptions. Request input and output parameters are assumed to be mapped to concepts from domain ontology. Therefore, when a service consumer wants to insert his request, an Ontology-Guided Interface is offered and the service consumer must select the desired terms he wants to use in his request from the list of terms provided in a pop-up by the interface.

Domain Ontologies: Service and request are described using relevant ontology concepts. Different domains may need different ontological representations. Therefore, to avoid the semantic heterogeneity due to the use of different concepts, we use a common ontological basis which contains comprehensive ontologies of different domains of Web services development. Input and output parameters of both request and service are defined using concepts from the same domain ontology.

OWL Public Ontologies: Several websites provide open public ontologies such as DAML Ontology Library[1], which contains more than 280 ontologies written in OWL or DAML+OIL. OWL public ontologies are used to create new ontologies or update already existing one in the domain ontologies repository. These public ontologies will be consulted periodically to develop or enrich the domain ontologies.

[1] www.daml.org/ontologies/

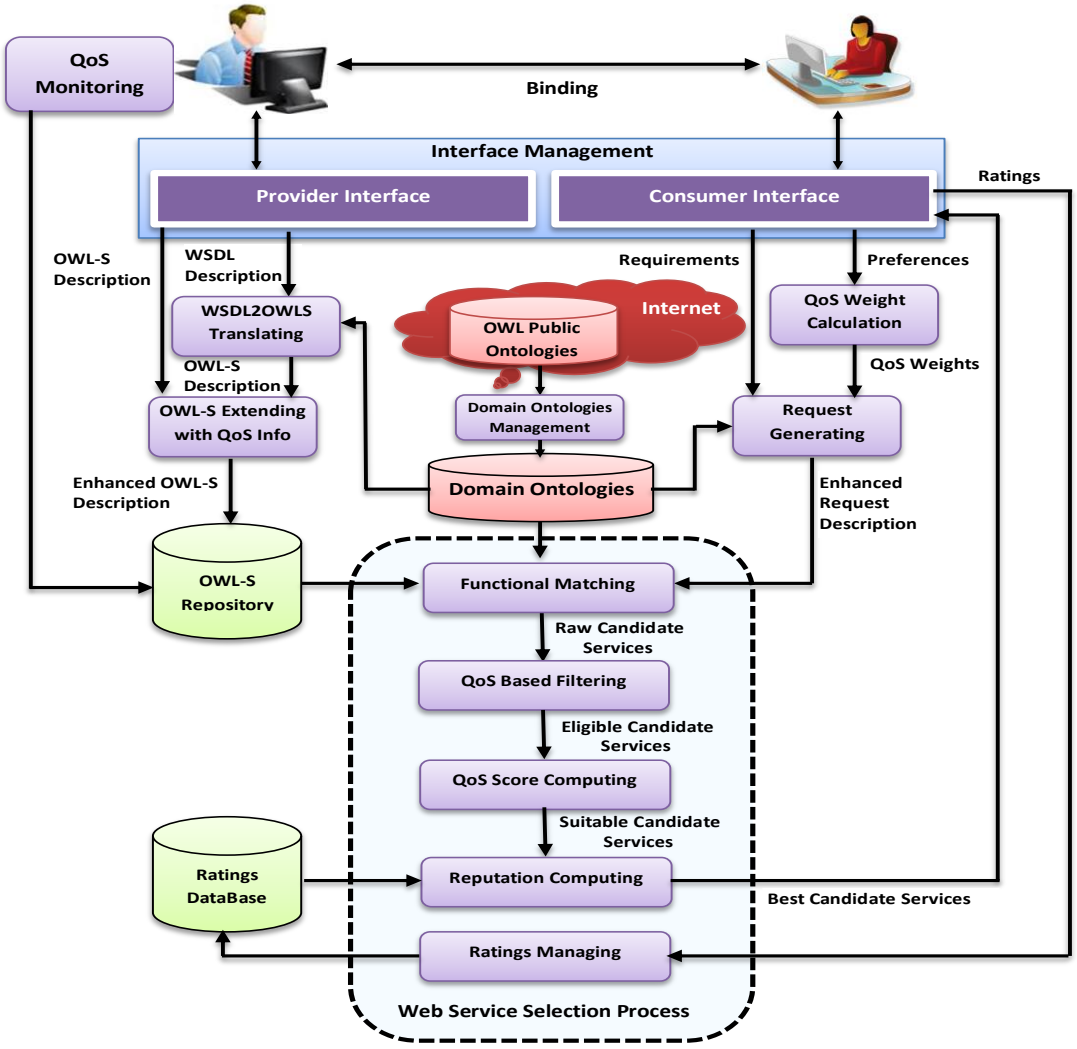

Figure 2. PrefWS3 architecture.

Domain Ontologies Management: This component takes charge of maintaining and updating domain ontologies with additional entities. The main challenges in updating domain ontologies are: 1) finding new information (concepts, relationships…), and 2) incorporating the new information in ontologies.

Functional Matching: The web service input and output parameters contain the underlying functional knowledge that is extracted for improving functional service discovery. The main concept of service discovery is semantic-based matching between requests and services. It establishes a mapping between the input of the request and the input of the service and a mapping between the output of the request and the output of the service.

QoS Based Filtering: Sometimes, the service consumer indicates that he refuses a Web service with a QoS having a value below or above a threshold specified in his query. QoS based filtering aims to filter out services that do not meet the service consumer preferences described as QoS constraints.

QoS Score Computing: The role of the QoS score computing step is to find the degree of quality of the candidate services after completing the functional matching through their QoS metric information and user preferences.

QoS Monitoring: The QoS monitoring mechanism aims to monitor and measure the QoS values of services in order to verify whether the measured values are in compliance with QoS values published by the Web service provider. QoS monitoring becomes the determining factors for customers to whether continue using the service or not [28].

Ratings Managing: Once the web service is selected, the service user should provide a rating score to show his satisfaction level of the invoked web service. The reputation mechanism enables service consumers to evaluate the credibility of web services they use, and takes into account the satisfaction criteria of each service consumer.

Ratings Database: User ratings are stored in an RDF triple store and kept in Ratings database.

Reputation Computing: The reputation of a service is a collective measure of the opinion of a community of users regarding their experience with the service [29]. It

is computed as an aggregation of users' feedbacks and kept in Ratings Database. Web services are ranked using their reputation and as a final step, best ranked candidate services are shown to the service consumer.

4 Web service and request model

Typically, Web services are described using functional and non-functional properties. Functional properties represent the description of the service functionalities. In our work, functional properties contain Service Name, Textual description, a set of Inputs and a set of Outputs. Non-functional properties represent the description of the service characteristics (e.g. QoS). Generally, QoS may cover a lot of attributes hosted by different roles. In this paper, we adopt three key attributes that the service customers mostly care about when they use a Web service. These are: Execution time, Execution price, and Reliability. Note that other QoS attributes can be applied to our system without fundamental modification.

A request signifies a service demand. A request description includes functional and non-functional requirements. The former describes the functional characteristic of the service demand, such as inputs and outputs. The latter mainly focuses on the customer's preferences, namely quality of service (QoS). In our work, a service consumer doesn't have to give the value of each desired QoS attribute; he should get instead the means to specify that a QoS attribute is more important than another one.

4.1 Translation from WSDL to OWL-S descriptions

The WSLD2OWLS translating component of PrefWS3 system translates WSDL files of the already existing Web Services into a semantic definition using OWL-S. This translation aims to add semantic annotations to Web Service specifications. In PrefWS3 system, we use only the OWL-S service profile in the discovery mechanism, so that the translator is responsible for translating WSDL files into OWL-S service profile files. Much research work has been done in mapping WSDL to OWL-S [30] [31] [32], but it is important to note here that until nowadays, the mapping process is not functioning fully automatically. The main raison is that the OWL-S description contains more information than the WSDL description. WSDL description provides only input and output information, while OWL-S description can provide inputs, outputs, preconditions and effects. Therefore this additional information must be set manually. In our work, we are limited to use only service name, textual description, inputs and outputs to functionally describe a web service. Because that information are already provided by WSDL description, the translator process is fully automatic.

The benefit of describing Web services in WSDL format is that WSDL is machine-readable, namely it can be parsed automatically. WSDL description contains service name, service textual description, types, inputs, outputs and binding. Based on the mapping process

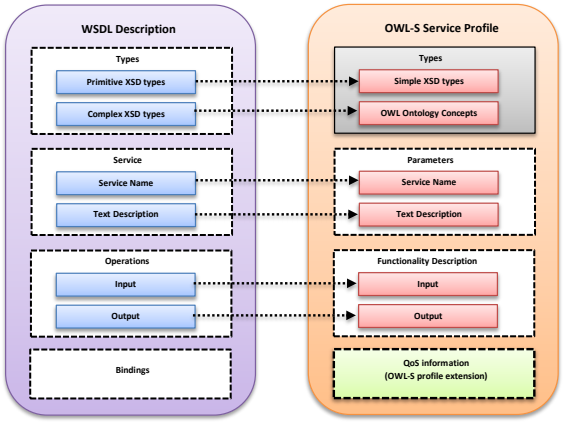

Figure 3: Translation from WSDL to OWL-S descriptions.

presented in [31], we can summarize our translator mechanism, as depicted in Figure 3, as follows:

- Service name in WSDL is translated into service name in OWL-S service profile.
- Service textual description in WSDL is translated into Service textual description in OWL-S service profile.
- Primitive XSD types in WSDL are translated into XSD simple types.
- Complex XSD types in WSDL are translated into OWL ontology concepts.
- Inputs and outputs of WSDL are mapping to OWL ontology concepts.

4.2 Embedding QoS properties in the OWL-S service profile

PrefWS3 system uses OWL-S service profile as a model for semantic annotation of Web service descriptions. However, OWL-S mainly focuses on describing functional aspects of a Web service and does not describe QoS aspects. Many approaches based on ontologies have been proposed for QoS [33] [34]. However, existing approaches are difficult for users to define their QoS based preferences. They usually assume that users could formulate their preferences easily and are accurately using the QoS languages.

Based on our previous work [9], we propose an ontology based OWL-S extension to add non-functional description, referred to as QoS, to Web service description. The new service profile model is depicted in Figure 4. In OWL-S service profile we use a set of ServiceParameter which has a name (serviceParameterName) and a value (sParameter). For the connection of OWL-S and QoS ontology, the QoSProperty is a subclass of OWL-S ServiceParameter, and QoSParameterName and qosParameter are subproperties of OWL-S ServiceParmaerterName and sParameter property respectively. This method is open to apply any QoS ontologies.

Each QoS property (QoSProperty) is defined by a name (qosParameterName) as a "String" and a set of characteristics (QoSCharacterisitic) that we describe as follows:

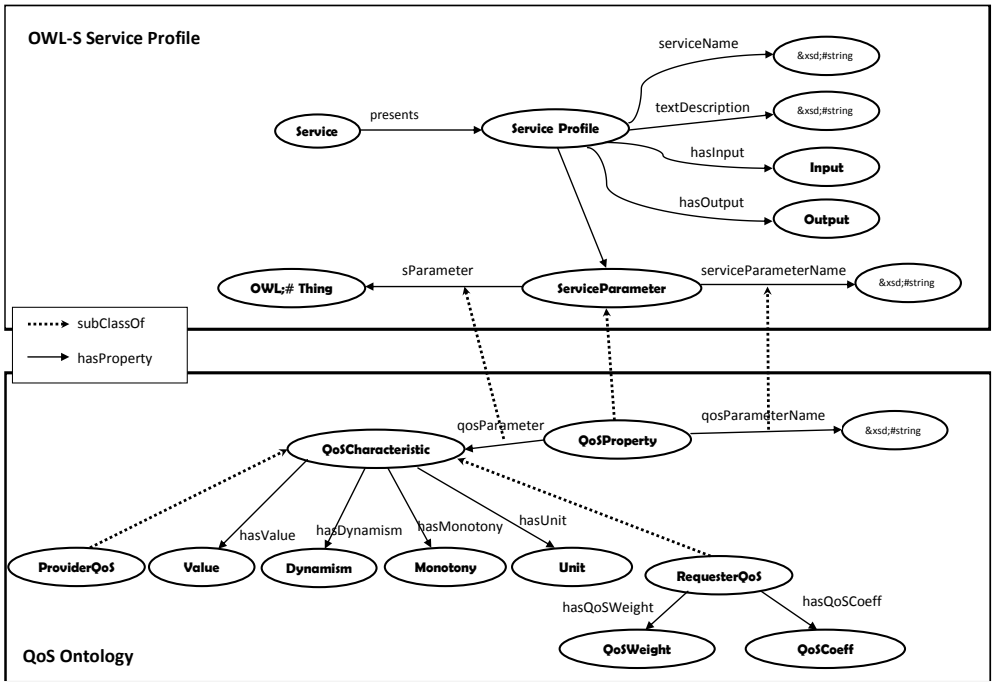

Figure. 4: OWL-S extension to support QoS.

- *Value:* Represents the value of a QoS property. From the provider viewpoint, this value represents the one of a QoS attribute of the provided service; but from the consumer viewpoint, it represents a threshold QoS value.
- *Monotony: This feature is used to distinguish between two types of QoS:*
  - Quality with increasing monotony (e.g, execution time). In this case, the QoS property value indicated by the service user represents the minimum value to be taken into account.
  - Quality with decreasing monotony (e.g, execution price."). In this case, the QoS property value indicated by the service user represents the maximum value to be taken into account.
- *Unit*: Each value of the QoS property is provided together with a measuring unit (e.g, Dollars, Seconds)
- *Dynamism*: We distinguish two different types of QoS: The static and dynamic. A static QoS is a quality whose value is known before the Web service execution (eg, the execution price). Dynamic QoS is a quality whose value is known only after the Web service execution (eg, execution time).
- *QoSWeight*: As described in previous section, the QoS weight allows specifying that a QoS property is more important than another one.
- *QoSCoeff*: This coefficient represents the degree of confidence that a customer has on his preference. The use of this coefficient will be detailed in subsection 6.1.

The proposed OWL-S extension is particularly useful for the different actors involved in the publication, the discovery and the invocation of web services, mainly service provider and service consumer (or user). Service provider can enter the services by filling properties values of the different service qualities (ProviderQoS). To facilitate this task, PrefWS3 displays the definition and comments on the quality property whose value must be entered. The service consumer can query the OWL-S extension ontology to find services that best meet their QoS requirements (RequesterQoS).

4.3 QoS weights calculating using AHP method

To help the service user on determining the weights according to their QoS preferences easily, the "QoS weight calculation" component of PrefWS3 uses a mechanism based on an Analytic Hierarchy Process (AHP) method which allows the calculation of weights only by a simple evaluation between two QoS attributes.

The Analytic Hierarchy Process, presented in [35], is a multi-criteria decision-making approach which can be used to solve complex decision problems. Basically, this approach involves the construction of a pair-wise comparison matrix where each element is rated against every other element by means of predefined scores (from 1 to 9) indicating their relative importance as shown in Table 1. These comparisons are used to obtain the weights of importance of the decision criteria. If the comparisons are not perfectly consistent, then it provides a mechanism for improving consistency.

In PrefWS3 system, the main steps for using the AHP method can be described as follows:

1. In the first step, we identify the criteria to be used by the method. As an illustration, we choose three QoS attributes as criteria, which are: execution time, execution price and availability.

2. In the second step, we establish the pairwise matrix based on service user preferences. By applying the AHP method, since we have three QoS attributes, a pairwise comparison matrix, containing nine elements, has been constructed. Suppose that the matrix, depicted in Table 2, represents the corresponding judgments with the pairwise comparisons.

Scales	Degree of preferences	Explanation
1	Equally	Two activities contribute equally to the objective
3	Moderately	Experience and judgment slightly favor one over the another
5	Strongly	Experience and judgment strongly or essentially favor one activity over another
7	Very strongly	An activity is strongly favored over another and its dominance is shown in practice
9	Extremely	The dominance of one over another is affirmed on the highest possible order
2, 4, 6, 8	Intermediate values	Used to represent compromises between the preferences in weights 1, 3, 5, 7 and 9
Reciprocals	Opposites	Used for inverse comparisons

Table 1: Pairwise comparison scale for AHP preferences. [35]

QoS attribute	Execution time	Availability	Execution price
Execution time	1	9	3
Availability	1/9	1	1/5
Execution price	1/3	5	1

Table 2: Example of a pairwise matrix.

3. In the third step, we calculate the weight of importance of each QoS attribute based on the pairwise comparison matrix and many normalization operations. The weighted values are calculated by Algorithm 1.

4. In the fourth step, we verify the consistency of the service user judgments. In the AHP method, judgments are considered to be adequately consistent if the corresponding consistency ratio (CR) is less than 0.1; otherwise it is necessary to review the subjective judgments. The CR is calculated as follows. First the consistency index (CI) needs to be calculated. This is done by algorithm 2. Next the consistency ratio CR is obtained by dividing the CI value by the Random index (RI) as given in Table 4 where *n* is the number of criteria.

When applying the algorithm on the above example of pairwise comparison matrix, we get the weights presented in Table 3.

Algorithm 1: Weights Calculation

Input: C: matrix n × n
// pairwise comparison matrix obtained in step 2.
Output: W: vector with size n // weights vector
Variables: P: matrix n × n initialized with 0 for each element.
S, W: vectors with size n initialized with 0 for each element.
Begin
1: **for** j := 1 to n **do**
  for i := 1 to n **do**
    S[j] ← S[j] + C[i][j];
2: **for** j := 1 to n **do**
  for i := 1 to n **do**
    P[i][j] ← C[i][j]/ S[j];
3: **for** i := 1 to n **do**
  for j := 1 to n **do**
    W[i] ← W[i] + P[i][j];
4: **for** i := 1 to n **do**
  W[i] ← W[i]/n;
End

QoS attribute	Execution time	Availability	Execution price
Weight	0.67	0.06	0.27

Table 3: Example of weight scores.

n	1	2	3	4	5	6	7	8	9
RI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45

Table 4: RI values for different values of n. [35]

When the algorithm 2 is applied to the previous judgment matrix, it can be verified that the following are derived: λ_{max} = 3.056, CI = 0.028, and CR = 0.048. The CR value is less than 0.10, so weights are accepted.

Cases	Concept A	Concept B	The role of concepts in request/web service	Relationship between Concepts in Domain Ontology	ConceptSim(A, B)
1 (line 1)	Location	Location	/	Location = Location	1
2 (line 3)	PhdStudent	Person	PhdStudent ∈R1.Inputs and Person ∈ S1.Inputs	PhdStudent < Person	1
3 (line 4)	AlgUniversity	University	AlgUniversity ∈ R1.Outputs and University ∈ S1.Outputs	AlgUniversity < University	0,8
4 (line 7)	University	AlgUniversity	University ∈ R2.Outputs and AlgUniversity ∈ S2.Outputs	AlgUniversity < University	1
5 (line 8)	Person	PhdStudent	Person ∈ R2.Inputs and PhdStudent ∈ S2.Inputs	PhdStudent < Person	0,6
6 (line 10)	PhdStudent	Employer	/	PhdStudent <> Employer	0,5
7 (line 11)	Person	University	/	Person ≠ University	0

Table 5: Example of conceptSim calculation.

5 Service functional matching

The main concept of service functional matching is semantic matching between request and web services, namely, inputs and outputs of the request are both matched with the ones of the web service. We consider that all inputs and outputs refer to concepts of domain ontology. In fact, matching inputs (outputs) of the request and the web service is nothing other than the matching of concepts associated to inputs (outputs). To calculate the similarity of two concepts A and B, we take into account two parameters. The first is the relationship between the two concepts in the domain ontology. The second is the role of concepts in the request and the web service, i.e, concepts are inputs or outputs.

Based on the relationship between the two concepts A and B in the domain ontology, we distinguish the following scenarios:

- A = B: The concepts A and B are the same or they are declared as equivalent classes.
- A < B: The concept A is a subclass of the concept B directly or indirectly.
- B < A: The concept B is a subclass of the concept A directly or indirectly.
- A <> B: The concept A does not have a parent/child relationship with the concept B, but both concepts have a parent concept C in common directly or indirectly.
- A≠B : Otherwise.

Based on the role of concepts in both request and web service, we think that an output in the request should not be considered as similar to a more generic output in the advertised service, while a request input could be considered as similar to a more generic advertised input. For example, if a user requests a web service that gives as an output the list of "Algerian universities", then the web service that gives as an output the list of all universities, cannot be considered as a suitable service because; it can return a set of "European universities"

Algorithm 2: CI Calculation

Inputs: C: matrix n × n // pairwise comparison matrix obtained in step 2**.**
W: vector with size n // weights vector obtained by Algorithm 1
Outputs: CI: float // Consistency Index
Variables: P: matrix n × n initialized with 0 for each element.
S: vector with size n initialized with 0 for each element.
λ_{max}: float.
Begin
1: **for** j := 1 to n **do**
  for i := 1 to n **do**
    P[i][j] ← C[i][j]*W[j];
2: **for** i := 1 to n **do**
  for j := 1 to n **do**
    S[i] ← S[i] + P[i][j];
3: **for** i := 1 to n **do**
  S[i] ← S[i]/W[i];
4: λ_{max} ← Max(S[1], S[2],….., S[n]);
5: CI ← (λ_{max} – n)/(n – 1);
End

that do not interest the user. We think also that an input in the advertised service should not be considered as similar to a more generic input in the request, while an output in the advertised service could be considered as such. For example, if a user requests a web service that takes as an input the ID of a student, then the Web service that takes as an input only the ID of a PHD student cannot be considered as a suitable service, because it ignores a much of the request's inputs.

To calculate the semantic similarity between two concepts A and B, we use the function ConceptSim(A, B). Our definition of this function is based on the constraints described above and on the information theoretic based measure presented in [36]. Semantic similarity is defined as the amount of common information that is shared between the concepts. Algorithm 3 gives the exact definition of the function ConceptSim(A, B), where:

- The concept A annotates an input/output of the request, while the concept B annotates an input/output of the Web Service.
- All inputs and outputs refer to concepts of the domain ontology, an example portion of which is shown in Figure 5.

Algorithm 3 : ConceptSim(A, B)

Begin
1: **if** A = B **then** ConceptSim(A, B) = 1
2: **else if** A < B **then**
3: **if** A, B are Inputs **then** ConceptSim(A,B)= 1
4: **else if** A, B are Outputs **then**
$$\text{ConceptSim(A, B)} = \frac{\text{Size(prop(B))}}{\text{Size(prop(A))}}$$ **endif**
5: **endif**
6: **else if** B < A **then**
7: **if** A, B are Outputs **then** ConceptSim(A,B) = 1
8: **else if** A, B are Inputs **then**
$$\text{ConceptSim (A, B)} = \frac{\text{Size(prop(A))}}{\text{Size(prop(B))}}$$ **endif**
9: **endif**
10: **else if** A <> B **then**
$$\text{ConceptSim(A, B)} = \frac{\text{Size(prop(A)}\cap\text{prop(B))}}{\text{Size(prop(A)}\cup\text{prop(B))}}$$
11: **else** ConceptSim (A, B) = 0 **endif**
12: **endif**
13: **endif**
14: **endif**
13: **return** ConceptSim (A, B).
End

- The function prop(C) denotes the set of properties of the concept C.
- The function Size(S) denotes the number of elements of the set S.
- If a concept A is a subclass of a concept B (A < B), then all properties of B are added to the properties of A (inheritance property).

Example: For illustration, let us take two requests (R1, R2) and two web services (S1, S2). All inputs and outputs refer to concepts of the domain ontology shown in Figure 5.

- R1: Inputs = { PhdStudent}, and
  Outputs = { Location, AlgUniversity }
- R2: Inputs = { GeographicArea, Person }, and
  Outputs = { University }
- S1: Inputs = { Person }, and
  Outputs = { Location, University }
- S2 : Inputs = { Location, PhStudent }, and
  Outputs = { AlgUniversity}

The different cases can be illustrated in Table 5.

After describing the semantic similarity between concepts, we give now the algorithm of inputs matching (algorithm 4). Where R.Inputs and S.Inputs denote the set of inputs in the request R and the set of inputs in the service S respectively, Card(E) denotes the cardinality of the set E, Sort(A) allow to sort the elements of the array A in descending order. In lines 1, 2, 3 and 4, the algorithm matches each request input with all Web service inputs, and keeps the best mapping for each request input. In lines 9, 10, 11 and 12, it distinguishes between the situation when the number of request inputs is less than the number of service inputs and when the inverse situation is presented. In the first case, we have a miss of information; therefore InputsSim value is decreased (line 10).

The outputs similarity given by OutputsSim(R.Outputs, S.Outputs) function is also calculated, by algorithm 5, in the same way as inputs similarity. But when the number of service outputs is less than the number of request outputs, the value of OutputsSim is decreased. Therefore we inverse line 10

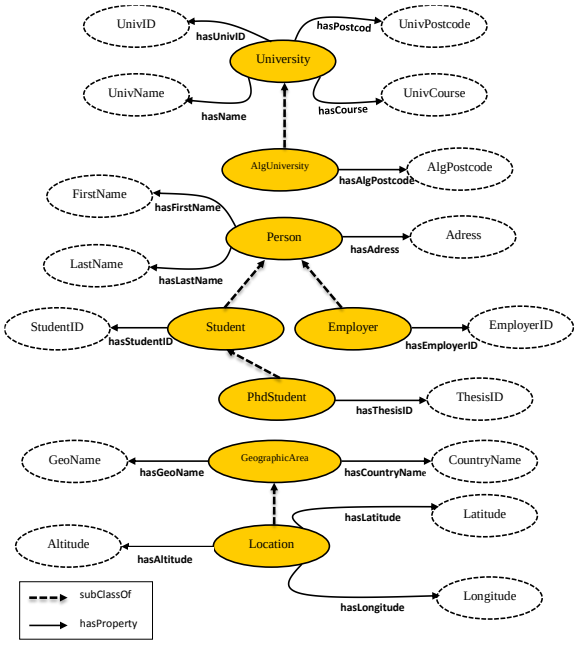

Figure 5: Part of simple Ontology.

with 12 and perform changes in variable names in the algorithm 3.

For example, let us calculate the Inputs and Outputs similarity between Req1 and WSer1 shown previously.
InputsSim = ConceptSim(PhdStudent, Person) = 1.

$$\text{OutputsSim} = \frac{\text{ConceptSim(Location,Location)} + \text{ConceptSim(AlgUniversity,University)}}{2} = \frac{1+0,8}{2} = 0,9.$$

After calculating inputs and outputs similarity, functional similarity can be calculated using Equation 1. Where weights w_1 and w_2 are real values between 0 and 1 and must sum to 1; they indicate the degree of confidence that the service consumer has in the input similarity and output similarity. By default, w_1 and w_2 are set to 0.5.

$$\text{FunctionalSim}(R, S) = w_1 * \text{InputsSim}(R.\text{Inputs}, S.\text{Inputs}) + w_2 * \text{OutputsSim}(R.\text{Outputs}, S.\text{Outputs}) \quad (1)$$

In the previous example, FunctionalSim(R1, S1)= 0.5*1 + 0.5*0.9= 0.95. This value indicates that R1 and S1 are semantically very close.

Algorithm 4 : InputsSim(R.Inputs, S.Inputs)

InSim: array of float; // initialized with 0 for each element
Begin
1: **foreach** e_1 in R.Inputs **do**
2: **foreach** e_2 in S.Inputs **do**
3: $InSim_i$ = Max($InSim_i$, ConceptSim(e_1, e_2));
4: **end for**
5: i = i + 1;
6: **end for**
7: *Sort*(InSim);
8: *m* = Card(R.Inputs) – Card(S.Inputs);
9: **if** m<0 **then**
10: InputsSim = $\frac{\sum_{j=1}^{Card(R.Inputs)} InSim_i}{Card(R.Inputs)} / (|m| + 1)$
11: **else**
12: InputsSim = $\frac{\sum_{j=1}^{Card(S.Inputs)} InSim_i}{Card(S.Inputs)}$
13: **end if**
14: **return** InputsSim
End

6 The QoS-based matching phase

6.1 QoS based services filtering

Sometimes, the service user indicates that he refuses a Web service with a QoS having a value below or above a threshold specified in his query. For example, a service consumer may want a service with an execution price not exceeding 100 units. So, candidate services which are over this threshold value will be eliminated.

This type of filtering is effective to meet user preferences, but suppose for the previous query, the discovery process has found a good Web service from the functional point of view but offers execution price equal to 101 units. This Web service will be ignored although 1unit may not make a difference to the user. In such case, we propose that when the user indicates a threshold for a QoS attribute, it associates a confidence coefficient "QoSCoeff". This coefficient represents the degree of confidence that the user has on the specified threshold. The value of this coefficient should be in the range [0, 1]. The value 1 means that the filtering algorithm must strictly observe the specified threshold. The value 0 means that the filtering algorithm must ignore this threshold. Therefore, our system uses algorithm 6 as a QoS-based services filtering algorithm. With this algorithm we can avoid the selection of web services that does not meet the service consumer preference.

Algorithm 6 takes as inputs a set of candidate web services and a set of QoS based constraints, thresholds (QoSConstraints.value) and confidence coefficient (QoSConstraints.QoScoeff), then filter out unwanted services taking into account that each QoS attribute can be monotonically increased or decreased.

For each Web service from the candidate Web services, we check the offered QoS properties to compare it with user constraints.

Line 9: If the QoS property is a positive quality (QoSCharacteristic.Monotony = "increase"), then multiply the value of the threshold by the coefficient to further decrease the threshold value.

For example, if the user indicates a threshold equal to 50 units for the execution time QoS property, with a confidence coefficient equal to 0.7, then all Web services with an execution time more than 50 * 0.7 = 35 units are maintained. The others are filtered out.

Line 11: If the QoS property is a negative quality (QoSCharacteristic.Monotony = "decrease"), then we divide the value of the threshold by the coefficient to further increase the threshold value.

For example, if the user indicates a threshold equal to

Algorithm 5 : OutputsSim(R.Outputs, S.Outputs)

OutSim: array of float; // initialized with 0 for each element
Begin
1: **foreach** e_1 in R.Outputs **do**
2: **foreach** e_2 in S.Outputs **do**
3: $OutSim_i$ = Max($OutSim_i$, ConceptSim(e_1, e_2));
4: **end for**
5: i = i + 1;
6: **end for**
7: *Sort*(OutSim);
8: *m* = Card(R.Outputs) – Card(S.Outputs);
9: **if** m<0 **then**
10: OutputsSim = $\frac{\sum_{j=1}^{Card(S.Outputs)} OutSim_i}{Card(S.Outputs)}$
11: **else**
12: OutputsSim = $\frac{\sum_{j=1}^{Card(R.Outputs)} OutSim_i}{Card(R.Outputs)} / (|m| + 1)$
13: **end if**
14: **return** OutputsSim
End

100 units for the execution price QoS property, with a confidence coefficient equal to 0.8, then all Web services with an execution price less than 100/0.8 = 125 units are maintained. The others are filtered out.

Algorithm 6: QoSServicesFiltering(CandidateServices, QoSConstraints)

```
Begin
1:  foreach service S in CandidateServices do
2:    foreach QoSParameter in S do
3:     Coeff := QoSConstraints.QoScoeff
4:     Mon:= QoSCharacteristic.Monotony
5:     SVal:= QoSCharacteristic.Value
6:     RVal:= QoSConstraints.Value
7:    if (Coeff < > 0) then
8:      if (QoSParameter.name = QoSConstraints.name)
       then
9:         if (Mon = "increase" ) and
              (SVal < (RVal × Coeff)) then
10:             FilterOut (S) from CandidateServices.
11:       elseif (Mon = "decrease" ) and
              (SVal > RVal / Coeff)) then
12:             FilterOut (S) from CandidateServices.
        endif.
       endif
      endfor
     endfor
End
```

6.2 QoS score computing

Each QoS value (qosValue) needs to be normalized to have a value in the range of 0 to 1. This step normalizes them in [0, 1] to guarantee they are evaluated by the same span. To normalize the QoS value, we take into account that each QoS attribute is monotonically increasing or decreasing.

	QoS attribute nature	qosMaxValue and qosMinValue
	monotonically increasing	qosMaxValue ≠ qosMinValue
Normalized QoS value	$1 - \frac{\text{qosMaxValue} - \text{qosValue}}{\text{qosMaxValue} - \text{qosMinValue}}$	
	monotonically increasing	qosMaxValue = qosMinValue
Normalized QoS value	1	
	monotonically decreasing	qosMaxValue ≠ qosMinValue
Normalized QoS value	$1 - \frac{\text{qosValue} - \text{qosMinValue}}{\text{qosMaxValue} - \text{qosMinValue}}$	
	monotonically decreasing	qosMaxValue = qosMinValue
Normalized QoS value	1	

Table 6: QoS value normalization.

Table 6 shows how to normalize QoS value, where qosMaxValue and qosMinValue values show the maximum and minimum values of the QoS attribute between all candidate services.Algorithm 6 takes as inputs a set of candidate services in "CandidateServices " and calculated QoS weights from a service user request and establishes the QoSServices matrix of QoS scores, and gives as output a vector QoSScore which contains the overall QoS score of each candidate Web service. QoSServices is a matrix where rows represent candidate Web services, and columns represent QoS attributes.

Algorithm 7: QoSScoreComputing(CandidateServices, QoSConstraints)

```
MtxServices: Matrix of float ;
QoSScore: Vector of float initialized by <0, 0,……, 0> ;
Begin
1:  foreach service S in CandidateServices do
     begin
2:    foreach QoSParameter in S do
       begin
3:      MtxServices[i, j] : =
         NormalizedValue(QoSCharacteristic.Value);
4:     QoSScore[i] := QoSScore[i] + (MtxServices[i, j] ×
         QoSConstraints.QoSWeight);
       j:= j +1;
      endfor
5:   i:= i +1;
     end for
End
```

In line 3, we calculate for each candidate Web service the normalized value of each QoS attribute.
In line 4, we calculate for each candidate Web service the overall QoS score which is the sum of each normalized QoS value multiplied by the weight given in the service user request.

6.3 QoS monitoring

The QoS monitoring process aims to monitor and measure the QoS values in order to verify whether the measured values comply with QoS values published by the Web service provider. As it is mentioned in Section 4.2, we distinguish two different types of QoS: The static and dynamic. The QoS monitoring process is interested in dynamic QoS monitoring, because QoS values are known only after the Web service execution.

The QoS monitoring in the field of Web services has been studied by many addressed (e.g., [[37] [38] [39] [40] just to name a few). The authors of [37] introduce a QoS model which covers various dimensions of QoS, i.e. availability, accessibility, performance, reliability, security, and regulatory, and propose metrics to enhance QoS measurement on the service side. They realized the monitoring of QoS dimensions above through a monitoring extension of Java system application server developed in Java EE 5.0. In [38], the authors present a Probe-based Observability Mechanism required for the monitoring of the web services that facilitates observation of internal execution details of the web services during testing and execution. The authors in [39] carry out a research to develop a monitoring method for web services response time. The method proposed in

this research is based on creating a proxy for connecting to the required Web service, and then calculating the Web services response time via the proxy. The work in [40] presents the Vienna Runtime Environment for Service-oriented Computing (VRESCo) that addresses some issues of current Web service technologies, with a special emphasis on service metadata, quality of service, service querying, dynamic binding and service mediation. The QoS monitoring is performed in their work to evaluate the framework through performance measurements on service querying, binding, mediation and invocation performances.

According to these studies, QoS monitoring can be performed into two approaches: (1) Client-side monitoring: the measurement of QoS is run on the client side [39], (2) Server-side monitoring: the measurement of QoS is run on the server side [37] [40]. On one hand,

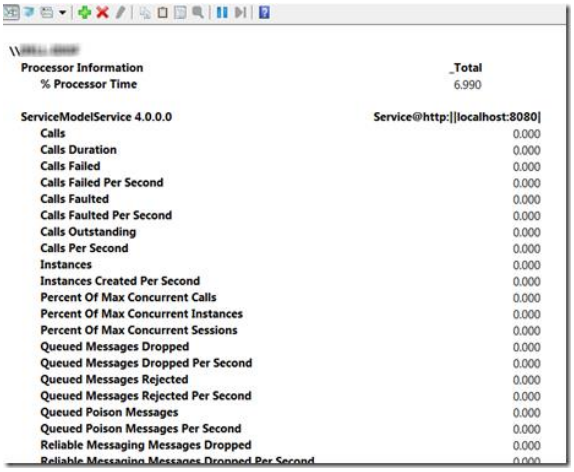

Figure 6: Windows Performance Counters: ServiceModelService Category.

client-side monitoring usually gives less accurate monitoring results and requires that clients must agree to install monitoring software which may not always be the case. But on the other hand, server-side monitoring is usually accurate but requires access to the actual service implementation which is not always possible.

In our work, we choose the use of a server-side monitoring mechanism, while ensuring that it does not affect existing implementations of the observed Web services. For this raison, our QoS monitoring mechanism is based on Windows Performance Counters (WPC) provided by Windows Communication Foundation (WCF) [41], which are part of the .NET Framework and offer a server-side QoS monitoring for Web services. Windows Performance Counters allow measuring the performance of Windows Communication Foundation Web services without altering any existing services.

WPC supports a rich set of counters that can be measured during the execution time of Web services. Performance counters are scoped to three different levels: Service, Endpoint and Operation. Each of these levels has performance counters to analyse the performance of a hosted WCF Web service. Service performance counters measure the service behaviour as a whole and can be used to diagnose the performance of the whole service. They can be found under the **ServiceModelService 4.0.0.0** performance object when viewed with Performance Monitor (Figure 6).

In our work, we focus on the following counters: "*Call Duration"* counter to measure the execution time, "*Calls Per Second"* counter to measure the number of a Web service invocations, and "*Failed Calls Per Second"* counter to measure the number of a Web service failures.

As depicted in figure 7, the way the QoS monitoring mechanism functions can be summarized as follows:

- Initially, the QoS monitor has to be installed on the service provider host. QoS monitor is itself a service which captures the performance counters of the monitored web services.
- Once installed, the QoS monitor has to be configured by setting the required parameters in the **Web.config** file. This configuration allows the operating system to attach the performance counters to the monitored web services.
- By default, the Windows Performance Counters are turned off because they could significantly increase the memory footprint of the WCF application. Performance counters can be enabled for the service from the diagnostics section of the **Web.config** file, as shown in the following sample configuration:

```
<configuration>
  <system.serviceModel>
      <diagnostics
   performanceCounters="All" />
  </system.serviceModel>
</configuration>
```

To specify the web service we want to monitor, we need to add its name in the services section of the Web.config file as follows:

```
<configuration>
  <system.serviceModel>
     <services>
        <service
     name="MonitoredServiceName" >
                  ……
          </service>
     </services>
   </system.serviceModel>
</configuration>
```

- Once started, the QoS monitor constantly continues reading the current values of the performance counters (Call Duration, Calls Per Second, Failed Calls Per Second) and transmits them to the QoS aggregator component of the PrefWS3 system. The QoS monitor sends sequentially, to the QoS aggregator, a SOAP message containing information about the service provider, the monitored service and the corresponding measured performances.

- When the QoS aggregator component receives the performance counters values sent by the QoS monitor, it aggregates these values to calculate the execution time and the reliability of the monitored Web service. The performance counter "Calls Duration" of the counter category "ServiceModelService 4.0.0.0" is used to calculate the execution time QoS, and the performance counters "Calls Per Second", "Failed Calls Per Second" of the same category are used to calculate the reliability QoS.
- Finally, the measured QoS values are transmitted to the decision maker component. This latter compares the measured QoS values with the corresponding QoS values published by the Web service provider in the OWL-S repository. If the QoS values published do not comply with the measured QoS values then the service provider will be punished. Several forms of punishments have been proposed. In our work, we propose to temporarily exclude the web service whose QoS are not real.

The QoS monitoring mechanism of the PrefWS3 system makes use of Windows Performance Counters, which are integrated into the operating system and thus, representing an easy way to QoS monitoring.

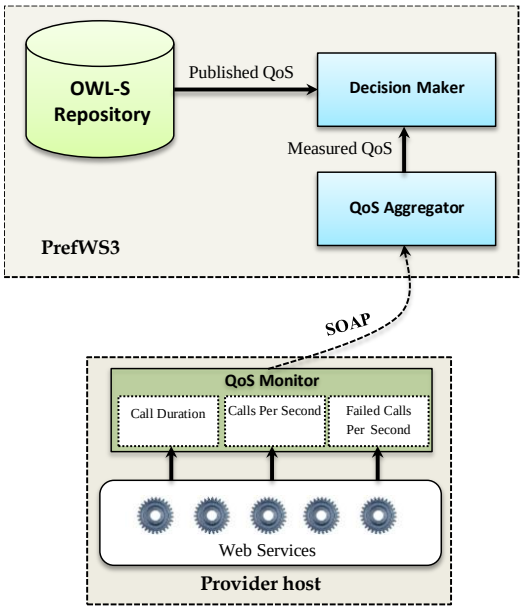

Figure 7. QoS monitoring mechanism.

7 Rating and reputation mechanism

Before paying the execution price of a Web service, the user is always looking to be sure of his choice. One of the mechanisms used to make the user have confidence in the selected web service is to give him the ratings of other users who have already used it. Once the web service is selected, the service user should provide a rating score to show the user satisfaction level of the invoked web service. A rating score is an integer number that ranges from 0 to 4, where the meaning of each value is as follows: 4: very satisfied, 3: satisfied, 2: neither satisfied or dissatisfied, 1: dissatisfied, 0: very dissatisfied.

In existing Rating-based approaches, the satisfaction criterion of the rater is unknown. Without knowing the intendment of the rater, it is almost impossible to make sense of a given rating. For example, a service user may give a high rating to a Web service because its execution time is small. If the execution time is not significant for a second service user, then the first service user's high rating will not be significant either. Hence, it is important to take into account the satisfaction criteria of each service user. This is done by giving a rate score for each QoS attribute of the used Web services.

The user ratings are stored in an RDF triple store. As user ratings refer to a given service request, each Rating instance contains the service user who performed the rating, the rated service, the rating date, and finally the rating scores (one rating score per QoS attribute). New ratings from the same user for the same service replace older ratings.

Over time, the qualities of a service can be changed by the service provider. In this case, old ratings are no longer representative. To address this problem, we give more importance to the recent ratings. This is done using Equation 2, where Δd is the number of days between the current date and the rating submission one. Figure 8 shows the evolution of the rating value over the time where the initial value equals 3.

$$\text{UpdatedRate}(\text{S. QoSProperty}) = \frac{Rate(S.\text{QoSProperty})}{\log_{10}(10+\Delta d)} \quad (2)$$

The reputation score of a service S within a single QoS attribute is computed as the average of all ratings the service receives from service users for this QoS attribute as indicated in Equation 3, where N is the number of ratings for the service S. Each rating score is normalized, as a monotonically increasing criterion, to have a value in the range of 0 to 1.

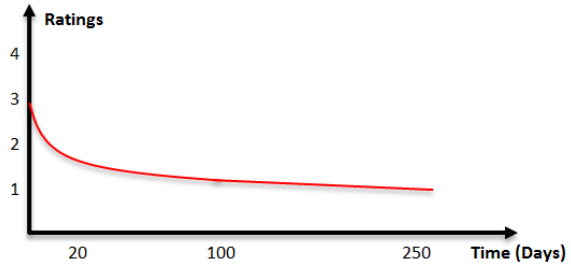

Figure 8: Example of the rating value evolution.

$$\text{ReputationScore}(\text{S. QoSProperty}) = \text{NormalizedValue}\left(\frac{\sum \text{UpdatedRate}(\text{S.QoSProperty})}{\text{N}}\right) \quad (3)$$

The reputation score of a service within multiple QoS attributes is computed, by Equation 4, as the weighted sum of the rating score of each quality attribute.

$$\text{OverallReputationScore}(\text{S}) = \frac{\sum \text{ReputationScore}(\text{S.QoSProperty}) * \text{weight}}{\sum \text{weight}} \quad (4)$$

8 Evaluation

In implementing PrefWS3, we use some software and tools. PrefWS3 is developed with Java under Eclipse IDE platform. PrefWS3 makes use the OWL-S API [42] for OWL-S files parsing. Jena 2.2 [43] is used for reasoning on OWL.

In order to evaluate the performance of our proposed semantic similarity algorithm which calculates the semantic similarity between a request and a web service, we compared it with two semantic matchmakers, the SAM architecture introduced in [15], and the BSA algorithm presented in [16].We use Book, Person and Printed Material ontology presented in [15], which is retrieved from "OWL-S Service Retrieval Test Collection version 2.1" available from the SemWebCentral Website[2]. In addition, we also used request and service definitions presented in the same work.

As shown in Figure 9, Book, Person and Printed Material ontology contains information on printed material classification and related concepts such as publishers, readers, authors, book types and several other concepts.

As the properties of the superclass are inherited by its subclasses, and in order to apply our algorithm, using the ontology described above, we assume that each subclass (or subconcept) in the ontology contains one more property than its superclass (superconcept). The request and the web services input/output parameters are given in Table 7. Request input concepts are Ordinary-Publisher, Novel, and Paper-Back. Request output concepts are Local-Author and Genre.

To demonstrate the value-added features of our semantic similarity algorithm, we present a test case between Request and Web Service 1 for input matching. The input parameters for Web Service 1, as shown in Table 7, are Publisher, ScienceFiction-Book. We calculated the semantic similarity using the ConceptSim function.

By applying the InputsSim algorithm, input concepts in both request and Web service 1 are matched as follows:

- Ordinary-Publisher → Publisher: ConceptSim = 1,

  since: Ordinary-Publisher < Publisher.

- Novel → ScienceFictionBook:

ConceptSim = $\frac{\text{Size}(\text{prop}(\text{Novel}) \cap \text{prop}(\text{ScienceFictionBook}))}{\text{Size}(\text{prop}(\text{Novel}) \cup \text{prop}(\text{ScienceFictionBook}))} = \frac{4}{6} = 0.666$, since:

Novel <> ScienceFictionBook.

- Paper-Back: No match

- Paper-Back is an extra input of a request, so it can be ignored and thus, the InputSim(Request, Web Service 1) = (1+0.666)/2 = 0.833

Results of the input-output similarity calculation of all services in the test case are listed in Table 8.

Service 2 is found to be the most similar to Request according to input matching, since it has the highest score for input matching of all the other classes. In fact, all the SAM, BSA, and PrefWS3 found this to be the best matched service in input matching with a score of 0.4388 by SAM, a score of 0.77 by BSA, and a score of 0.833 by PrefWS3. However, SAM, BSA, and PrefWS3 found the Service 2 has the weakest match outputs with scores of 0.01447, 0.012, and 0.125 respectively.

On the other hand, in PrefWS3, matching Request and Service 5 should give the highest score according to output matching since {Genre → Genre : ConceptSim = 1} and {Local-Author → Publisher : ConceptSim = 0.25}. Both SAM and BSA found this to be the best matched service for output matching and scored it as 1.00018 by SAM, and 1.2565 by BSA.

Furthermore, SAM and BSA found that Service 3 has the weakest match for inputs, so this places it the latest in the rankings, which was also found as unrelated and scored as 0.541 by PrefWS3.

All the SAM, BSA, and PrefWS3 found that Service 3 and Service 4 have the same output matching scores. Thus, for Service 3 and Service 4, BSA orders the results according to the maximum value of input scores, whereas SAM uses a random selection. Finally, both the BSA and PrefWS3 found the order of total score to be:

Service 5 > Service 1 > Service 4 > Service 2 > Service 3.

The results reveal that, in both BSA and PrefWS3 systems, Service 5 has the highest total score considering both input and output matching, and Service 2 has the lowest total score. As a conclusion, comparing the results given by PrefWS3 with those given by SAM and BSA, we note that PrefWS3 offers good results but with less calculation, and therefore less time.

9 Conclusion

In this article, we introduce a semantic web services discovery and selection system (PrefWS3). An advanced feature of PrefWS3 is that it performs the service discovery and selection based on the matching level of the service advertisements with the user requests in terms of both functional and non-functional parameters. PrefWS3 is considered to be a user-centric system which helps and guides users on formulating their requirements and preferences, and hence, allows to free consumers from time consuming human computer interactions and Web search. Additionally, PrefWS3 uses a translator to translate WSDL files into OWL-S and provides semantically enriched description. As a result, enhancing web services with a semantic description of their functionality will further improve their discovery and selection. PrefWS3 uses an efficient semantic-based

[2] http://projects.semwebcentral.org

matching mechanism which calculates the semantic

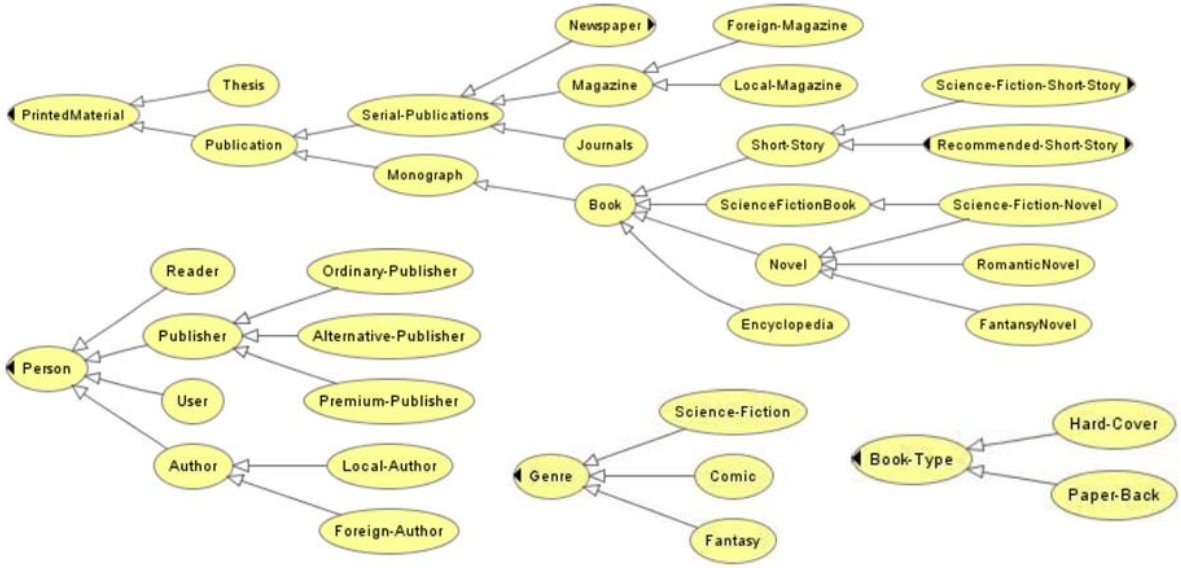

Figure 9: Book, Person and Printed Material ontology section. [15]

Request/Service Name	Inputs	Outputs
Request	Ordinary-Publisher, Novel, Paper-Back	Local-Author, Genre
Web Service 1	Publisher, ScienceFictionBook	Author, Price
Web Service 2	Book, Alternative- Publisher, Book-Type	Publisher, Price, Date
Web Service 3	FantasyNovel, Author	Price, Comic
Web Service 4	Newspaper, Book-Type, Person	Review, Fantasy
Web Service 5	Publication, Book-Type, Reader	Genre, Publisher

Table 7: Request and Services parameters.

Service name	Scores of SAM			Scores of BSA			Scores of PrefWS3		
	Input Sim Score	Output Sim Score	Total Score	Input Sim Score	Output Sim Score	Total Score	InputSim Score	Output Sim Score	Total Score
Service 1	0.35964	0.12229	0.21723	0.640	0.8571	0.7485	0.833	0.5	0.666
Service 2	0.4388	0.01447	0.27771	0.77	0.012	0.391	0.833	0.125	0.479
Service 3	0.18026	0.17033	0.08078	0.47	0.5076	0.4888	0.541	0.5	0.520
Service 4	0.23636	0.12229	0.69465	0.5321	0.5076	0.5198	0.761	0.5	0.630
Service 5	0.31718	1.00018	0.20024	0.575	1.2565	0.9157	0.75	0.625	0.687

Table 8: Comparison of PrefWS3, SAM, and BSA based on input/output parameter matching.

similarity between the request and the web service based on the concepts position in the ontology, the common properties between concepts, and also, either concept has annotated an input/output request parameter or an input/output web service parameter. Furthermore, PrefWS3 includes a QoS-aware process and provides a reputation mechanism that enables service users to evaluate the credibility of the web services they use, and takes into account the satisfaction criteria of each service user. In order to evaluate the effectiveness of our system, the results of a comparison of the PrefWS3 and some other published approaches (BSA and SAM) have been presented.

As future directions, we plan to incorporate the Web services composition into PrefWS3 in order to make it more practical in real-world applications. To this end, two main questions need to be asked:

1) How to combine Web services in a suitable way to fulfil the user request?

To answer this question, several approaches have been proposed such as: Constraint based composition, Business rule driven composition, AI Planning based composition, Context information based composition, Process based composition, and Model and aspect driven composition [44]. AI Planning approach has become interesting due to the maturity that the planning area has achieved in AI. We decide to extend our PrefWS3 system to support service composition by combining semantic matching and an AI planning technique. We focus on functional input and output parameters of Web services. The latter are respectively the preconditions and the effects in the planning context. Web service composition is then viewed as an AI planning based composition of semantic relationships between Web service parameters. To this end, we intend to adapt the functional matching mechanism of the PrefWS3 system to support semantic similarities between input and output parameters, and add a composition component that implements an AI planning technique.

2) How to select the best composition among a set of candidates that fulfil the same user request?

It is possible that the composition mechanism generates multiple composite services fulfilling the user request. In that case, the composite services are evaluated and ranked along the non-functional parameters such as QoS and user constraints, and the best composite service is the one which is ranked on top. Selecting a composite service that satisfies user constraints and preferences can be viewed as a Constraint Satisfaction Problem (CSP). To this end, we intend to formulate QoS based web service composition as a CSP, and adapt our QoS computing mechanism to compute the quality of a composite service when it is given the QoS of its underlying services.

References

[1] Averbakh. A, Krause. D and Skoutas. D. (2009). Exploiting User Feedback to Improve Semantic Web Service Discovery. *8th International Semantic Web Conference*. LNCS, Vol.5823, pp.33-48.

[2] Garofalakis. J, Panagis. Y, Sakkopoulos. E and Tsakalidis. A. (2006). Contemporary web service discovery mechanisms. *Journal of Web Engineering*. Vol.5, No.3, pp.265-290.

[3] Hyunkyung. Y. P and TaeDong. L. (2013). Ontology based keyword dictionary server for semantic service discovery. *IEEE Third International Conference on Consumer Electronics*. pp. 295 – 298.

[4] Paliwal. A.V, Shafiq. B, Vaidya. J, Hui. X and Adam. N. (2012). Semantics-Based Automated Service Discovery. *IEEE Transactions on Services Computing*. Vol.5, No.2, pp.260 – 275.

[5] Kopecky. J, Vitvar. T, Bournez. C and Farrell. J . (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Comput. Vol.1, No.11, pp.60-67.

[6] Roman. D, Keller. U, Lausen. H, de Bruijn. J, Lara. R, Stollberg. M, Polleres. A, Feier. C, Bussler. C and Fensel. D. (2005). Web Service Modeling Ontology. *Journal of Applied Ontology*. Vol.1, pp.77-106.

[7] Martin. D et al. (2004). OWL-S: Semantic Markup for Web Services. W3C. From http://www.w3.org/Submission/OWL-S/.

[8] Dasgupta. S, Aroor. A, Shen. F and Lee. Y. (2014). SMARTSPACE: Multiagent Based Distributed Platform for Semantic Service Discovery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. Vol.44, No.7, pp.805- 821.

[9] Benaboud. R, Maamri. R and Sahnoun. Z. (2013). Agents and owl-s based semantic web service discovery with user preference support. *Int. Journal of Web & Semantic Technology*. Vol.4, No.2, pp.57-75.

[10] Paolucci. M, Kawamura. T, Payne. T and Sycara. K. (2002). Semantic matching of web services capabilities. *Proceedings of the 1st International Semantic Web Conference (ISWC)*, Springer-Verlag. pp.333–347.

[11] Dong. H, Hussain. F and Chang. E. (2012). Semantic Web Service matchmakers: state of the art and challenges. *Concurrency and Computation: Practice and Experience*. Vol.25, No.5, pp.961-988.

[12] Klusch. M and Kapahnke. P. (2010). iSeM: Approximated Reasoning for Adaptive Hybrid Selection of Semantic Services. *In: Aroyo L, Antoniou G, Hyvönen E, ten Teije A, Stuckenschmidt H, et al.., editors. The Semantic Web: Research and Applications*. Springer Berlin. pp.30–44.

[13] Kiefer. C and Bernstein. A. (2008). The Creation and Evaluation of iSPARQL Strategies for Matchmaking. *In: Bechhofer S, Hauswirth M, Hoffmann J, Koubarakis M, editors. The Semantic Web: Research and Applications.* Springer Berlin. pp.463–477.

[14] Klusch. M and Kapahnke. P. (2012). Adaptive Signature-Based Semantic Selection of Services with OWLS-MX3. *Multiagent and Grid Systems*. Vol.8, No.1,pp.69–82.

[15] Erdem. S.I and Ayse. B. B. (2008). SAM: Semantic Advanced Matchmaker. *R. Nayak et al. (Eds.): Evolution of the Web in Artificial Intel. Environ*. Vol.130, pp.163–190.

[16] Çelik. D and Elçi. A. (2013). A broker-based semantic agent for discovering Semantic Web services through process similarity matching and equivalence considering quality of service. *Science China Information Sciences*. Vol.56, No.1, pp.1-24.

[17] De Renzis. A, Garriga. M, Flores. A and Cechich. A. (2016). Case-based Reasoning for Web Service

Discovery and Selection. Electronic Notes in Theoretical Computer Science. Vol.321, pp.89–112.

[18] Kolodner. J. (1993). Case-Based Reasoning, Morgan Kaufmann Publishers, Inc.

[19] Sakkopoulos. E, Kanellopoulos. D and Tsakalidis. A.(2006). Semantic mining and web service discovery techniques for media resources management. *Int. Journal of Metadata, Semantics and Ontologies*, Vol.1, No.1, pp.66-75.

[20] Xu. Z, Martin. P, Powley. W and Zulkernine. F. (2007). Reputation-Enhanced QoS-based Web Service Discovery. *In Proceedings of the International Conference on Web Services*. pp.249 – 256.

[21] Mobedpour. D and Ding. C. (2013). User-centered design of a QoS-based web service selection system. *Service Oriented Computing and Applications*. Vol.7, No.2, pp.117-127.

[22] Iordache. R and Moldoveanu. F. (2014). QoS-Aware Web Service Semantic Selection Based on Preferences. *Procedia Engineering.* Vol.69, pp.1152–1161.

[23] Chouiref. Z, Belkhir. A, Benouaret. K and Hadjali. A. (2016). A fuzzy framework for efficient user-centric Web service selection. Applied Soft Computing, Vol.41, pp.51-65.

[24] Chang. C and Kuo. C. (2013). A Web Service Selection Mechanism Based on User Ratings and Collaborative Filtering. *Smart Innovation, Systems and Technologies.* Springer-Verlag Berlin Heidelberg. Vol.20, pp.439-449.

[25] Nguyen. H. T, Zhao. W and Yang. J. (2010). A trust and reputation model based on bayesian network for web services. *IEEE International Conference on Web Services*. pp.251-258.

[26] Vu. L, Hauswirth. M and Aberer. K. (2005). QoS-based Service Selection and Ranking with Trust and Reputation Management. *Proceedings of the Confederated international conference on the Move to Meaningful Internet Systems*. pp.466-483.

[27] Grozavu. A, Pleşcan. S and Mărgărint. C. (2011). Comparative Methods for the Evaluation of The Natural Risk Factors' Importance. *Present Environment and Sustainable Development*. Vol.5, No.1, pp.41–46.

[28] Zadeh. M, Seyyedi. M. (2010). Qos monitoring for web services by time series forecasting. *3rd IEEE international conference on computer science and information technology (ICCSIT)*. pp.659–663.

[29] Limam. N and Boutaba. R. (2008). QoS and Reputation-aware Service Selection. *IEEE on Network Operations and Management Symposium*. pp.403-410.

[30] Heß. A, Johnston. E and Kushmerick. N. (2004). ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services. *In: Proceedings of the 3rd International Semantic Web Conference (ISWC)*. pp.320-334.

[31] Farrag. T, Saleh. A and Ali. H. (2013). Towards SWSs Discovery: Mapping from WSDL to OWL-S Based on Ontology Search and Standardization Engine. *IEEE Transactions on Knowledge and Data Engineering.* Vol.25, No.5, pp.1135-1147.

[32] Ashraf. B. (2014). Fast Mapping Algorithm from WSDL to OWL-S. *I.J. Information Technology and Computer Science*. Vol.6, No.9, pp.24-31.

[33] Lin. L, Kai. S and Sen. S. (2008). Ontology-based QoS-Aware Support for Semantic Web Services. *Technical Report at Beijing University of Posts and Telecommunications*.

[34] Zhang. Y, Huang. H, Yang. D, Zhang. H, Chao. H and Huang. Y. (2009). Bring QoS to P2P-based semantic service discovery for the Universal Network. *Journal Personal and Ubiquitous Computing*. Vol.13, No.7, pp.471–477.

[35] Saaty. T. (1995). Decision Making for Leaders. RWS Publications.

[36] [36] Lin. D. (1998). An information-theoretic definition of similarity. *In Proceedings of International Conference on Machine Learning*. pp.296-304.

[37] Artaiam. N and Senivongse. T. (2008). Enhancing service-side qos monitoring for web services. *In SNPD '08: Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. IEEE Computer Society. pp.765-770.

[38] Saxena. N, Goel. A and Singh. D. (2009). A probe-based observability mechanism for monitoring of web services. *Int J Recent Trends Eng*. Vol.1, No.1, pp.600–602.

[39] Asadollah. S. A and Thiam. K. C. (2011). Web service response time monitoring: architecture and validation. Theoretical and Mathematical Foundations of Computer Science. Vol.164, pp.276–282.

[40] Michlmayr. A, Rosenberg. F, Leitner. P and Dustdar. S. (2010). End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo. *IEEE Trans Serv Comput.* Vol.3, No.3, pp.193–205.

[41] Peiris. C, Mulder. D, Bahree. A, Chopra. A, Cicoria. S and Pathak. N. (2007). Pro WCF: Practical Microsoft SOA Implementation". Apress.

[42] Hewlett-Packard Development Company. (2001). Jena RDF API. from: http://www.hpl.hp.com/ semweb/jena.htm

[43] Mindswap-Maryland Information and Network Dynamics Lab. (2004). Semantic Web agents project: OWL-S Java API. from: http:// www.mindswap.org/2004/owl-s/api/index.shtml

[44] D'Mello. D. N, Ananthanarayana. V. S and Salian. S. (2011). A Review of Dynamic Web Service Composition Techniques. *First International Conference on Computer Science and Information Technology*. Springer Berlin Heidelberg. pp 85-97.