S^2R: A Semantic Web service Similarity and Ranking Approach

Amal Alhosban, Khayyam Hashmi, Zaki Malik
Wayne State University
and
Brahim Medjahed
University of Michigan - Dearborn

Service Oriented Architectures (SOAs) enable the automatic creation of business applications from independently developed and deployed services. Mechanisms are thus needed to select these service components that meet or exceed the functional and non-functional requirements of SOAs. The primary objective of service selection in SOAs can be viewed as a maximization of an application-specific utility function that matches the constraints of the service requester against the capabilities and offerings of the service provider(s). In this paper, we propose such an approach that computes the match between service requests and offerings, based on their functional and non-functional properties in an efficient manner (in terms of space and time). We compare our approach with similar existing approaches to its show applicability and performance.

Keywords: Composite services, Matching, Semantic Web, Service-oriented architecture, Similarity.

1. INTRODUCTION

Due to the competitive and fast growing nature of today's business climate, most organizations are automating their business processes for service and operation delivery. In this respect, service-oriented computing (SOC) has become a main trend in software engineering that exploits Web services as fundamental elements for developing on-demand applications. Web services are self-described, self-contained and platform-independent computational elements that can be published, discovered, and composed using standard protocols, to build applications across various platforms and organizations in a dynamic manner. With the increasing agreement on the functional aspects of Web services, such as using WSDL for service description, SOAP for communication and WS-BPEL for composing Web services etc., the research interest is shifting towards the non-functional aspects of Web services [Papazoglou et al. 2010].

Developers can now add descriptions (using standards such as OWL-S) to their Web services to define and advertise the non-functional aspects of services (including input, output, pre-condition, post-condition and functions), thereby facilitating automated discovery, invocation and inter-operation. However, the first step in this process is to 'resolve' the consumer request against prospective Web services, so that the most appropriate component could be selected [Alhosban et al. 2011]. The expected availability of a large number of highly specialized component services, means that it would be increasingly challenging to find the most suitable service(s) in a reasonable amount of time [Nepal et al. 2010]. Moreover, some Web services may not be able to satisfy consumer requests individually, and hence need to be integrated with other Web services to provide the desired functionality. This adds to the complexity of an already challenging problem [Bouguettaya et al. 2008].

In this paper, we present a novel approach (defined S^2R: Semantics-based Similarity and Ranking) for Web service selection. S^2R is divided into three levels. In the first level, we filter the

Authors' address: Department of Computer Science, Wayne State University, 5057 Woodward, Suite 3010, Detroit, MI 48202. Email: {ahusban, Khayyam, zaki}@wayne.edu
Department of Computer & Information Science, The University of Michigan - Dearborn, 4901 Evergreen Road, Dearborn, MI 48128. Email: Brahim@umich.edu

available Web services under a specific category based on their functional properties such as input, output and operations. In the second level, we further reduce the service search space based on non-functional properties, such as Quality of Service (QoS) parameters [Alhosban et al. 2011]. Once a reduced pool of similar Web services is obtained, we rank them based on their utility value (in the third level).

The utility value is calculated using a utility function which allows stakeholders to ascribe a value to the usefulness of the overall system as a function of several QoS attributes such as response time, availability, cost, reliability, etc. according to their preferences [Nepal et al. 2010] [Menascé and Dubey 2007] [Bergmann et al. 2001] [Li et al. 2012] [Yao et al. 2010]. Using utility function, S^2R filters Web services at each level so that more costly operations (e.g., reputation calculations) are applied on a reduced number of candidate services to shorten the time and space complexity of this search process. Moreover, since service selection is an on-demand process, we apply the S^2R filters on run time.

The rest of the paper is organized as follows. Section 2 provides a motivating scenario which is used to illustrate the S^2R approach further in the paper, while Section 3 lists the related works. Section 4 describes our approach in detail. Section 5 presents the experiments, Finally, Section 6 concludes the paper and provides directions for future work.

2. MOTIVATION

In this section, we present an example scenario to motivate the problem and associated solution. Assume a travel planning system that is based on a service-oriented architecture (Figure 1.). The company provides travel planning services that include hotel booking, flight reservation, and car rental. In addition to these reservation services, the system also provides an insurance service for the entire trip or individual travel components.

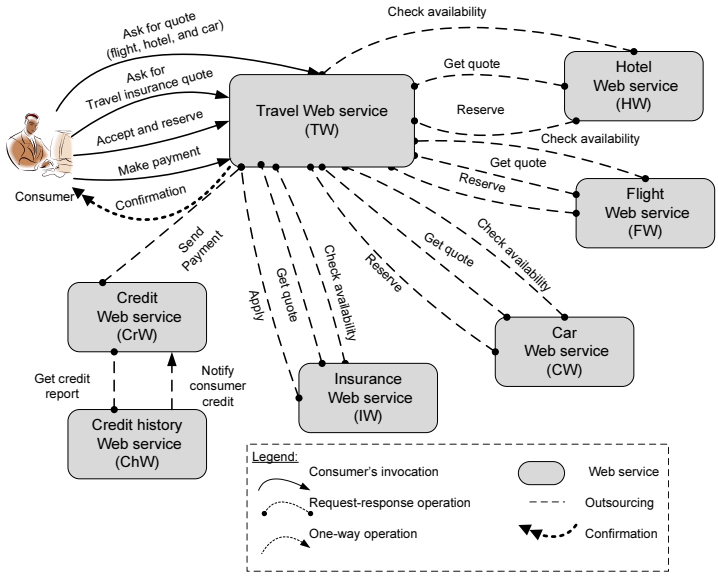


Figure. 1: Example Scenario: A Travel Reservation System

The main Web service for the system is called Travel Web service (TW) with major operations: `Check_Availability`, `Get_Quote`, `Reserve`, `Apply`, and `Send_Payment`. TW does not implement all these functionalities by itself, rather it outsources some of the functionality to other component Web services. In Figure 1. we can see that component Web services (outsourced Web services) include: Hotel Web service (HW), Flight Web service (FW), Car Web service (CW),

Insurance Web service (IW), and Credit Web service (CrW). The consumer invokes TW through the `Get_Quote` operation by providing the travel date, departure and arriving city information. To get the quote, TW should interact with other services (i.e., HW, FW, and CW) by checking the availability for the required dates and cities using operation `Check_Availability`. TW then requests quotes for the available reservations through the operation `Get_Quote`. Upon receiving individual quotes from component services, TW aggregates these quotes and sends them to the consumer. At this point in the reservation process, the consumer also has the option of buying travel insurance (which TW outsources to IW). If the consumer accepts the quote, the payment process starts. TW outsources CrW to process the credit payments. CrW in turn outsources the consumer's credit check process to ChW. If the consumer's credit meets the credit score requirements, then TW makes a reservation with (HW, FW, and CW) and starts the insurance process (if consumer wills). Finally, TW notifies the consumer with the confirmation number (for flight, hotel and rental car) and sends the receipt. TW may run into some issues when it is trying to formulate this solution by outsourcing functionalities to component services. First of all, how would TW calculate the functional equivalence of two or more similar services, e.g., when TW is looking for a flight Web service, the first step is to find all the Web services that provide this functionality (i.e., resolve both syntactic and semantic equivalence). Even if TW is able to find functionally similar Web services for flight Web service, they may have different non-functional (QoS) properties (such as service A may have a response time of 3ms and service B may take 7ms to respond to user requests). Hence TW needs to differentiate among the candidate services based on the value (utility) they add to the composition. The main motivation behind S^{2}R is to solve the above mentioned issues while reducing the time and space complexity of this (services) search process. We believe that an efficient solution to the service selection problem is also paramount in reducing fault recovery time in SOAs, for cases where a faulty service needs to be replaced by a 'similar' one [Alhosban et al. 2011].

3. RELATED WORK

In this section, we provide a brief overview of some of the related literature. Several methods have been proposed to deal with the Web service matching problem. The technique in [Xia and Yoshida 2007] uses two stage assessment. In the first stage all service belonging to a specific category are gathered. The second stage consisits of finding similarity among these services based on input, output, conditions and effects. LARKS [Sycara et al. 1999] defines five techniques for service matchmaking: context matching, profile comparison, similarity matching, signature matching, and constraint matching. Matching services to requests is performed by using any combination of the above techniques. The ATLAS matchmaker [Paolucci and Wagner 2006] defines two methods for comparing service capabilities described in DAML-S. The first method compares functional attributes to check whether advertisements support the required type of service or if it delivers sufficient quality of service. The second method compares the functional capabilities of Web services in terms of inputs and outputs. Anamika [Chakraborty et al. 2002] presents a service matching technique for pervasive computing environments. Service descriptions are provided in DAMLS. They also include platform specific information such as processor type, speed, and memory availability. The composition manager uses a semantic service discovery mechanism to select participant services. RACER [Li and Horrocks 2003] adopts techniques from knowledge representation to match DAML-S service capabilities. In particular, it defines a description logic (DL) reasoner; advertisements and requests are represented in DL notations.

Another DAML-S based matchmaker implementation is KarmaSIM [Narayanan and McIlraith 2002] where DAML-S descriptions are described in terms of a first-order logic language (predicates) and then converted to Petri-nets where the composition can be simulated, evaluated and performed. Context-based matching (CBM) has been proposed in [Medjahed and Atif 2007], the matching process is performed via peer-to-peer interactions between a context-based matching engine, CPAs and community services. A service consumer sends a matching request to context-

based matching engine which sends a sub request to the communities and compares the consumer requirement with each community members. Then the context-based matching engine finds the intersection between the matching set from each community. The communities have been created based on the policies inside the Web services. The problem in this technique is that the number of comparisons will be high if the same Web service exists in all communities (i.e., the Web service includes all policies that the consumer has requested). Circular context-based (CCB) has been proposed in [Segev 2008], the technique compares context extracted from each Web service based on its WSDL description to with other Web services' textual description context. The second stage consists of finding the context overlapp among the Web service through parsing WSDL file. Other service matching techniques are also presented in [Baïna et al. 2001] [Heuvel et al. 2001] [Mecella et al. 2001]. However, these techniques mostly focus on syntactic comparison among attributes of Web services.

Since we use contextual information of Web services, we position our work with existing context-oriented Web service frameworks. Several context-aware approaches have recently been proposed to enhance Web service discovery and composition mechanisms. Context attributes [Lee and Helal 2003] proposes a context-aware service discovery technique for mobile environments. It defines the context of a Web service as a set of attributes included in the service description. Examples of context attributes include user location and network bandwidth. The discovery engine first lookups for Web services based on traditional criteria (e.g., service category in UDDI). Then, it reduces the qualified services to be returned to clients through context attribute evaluation. This approach uses contextual information for service discovery not for service composition. Additionally, it focuses on client-related contextual information. It does not seem to consider provider-related context which is important for Web service composition. Finally, the definition of context is limited to some attributes added to service descriptions. We adopt a more generic definition of Web service context through an ontology-based categorization of contextual information. Contextualization is proposed at the Web service deployment, composition and conciliation or matching levels in [Maamar et al. 2006]. The description of contexts is assumed to occur along three categories: profile, process model, and grounding. The profile describes the arguments and capabilities of a context. The process model suggests how context collects raw data from sensors and detects changes, that need to be submitted to the Web service. Finally, the grounding defines the bindings (protocol, input/output messages, etc.) that make context accessible to a Web service. The authors did not however mention how relevant contexts are elicited in a service matchmaking process.

4. S^2R ARCHITECTURE

In this section, we present the architecture details of S^2R: a semantic Web service similarity and ranking approach. Generally, similarity measurement consists of two components: syntactic and semantic. In syntactic similarity, we look for the similarity between data items where value of this similarity usually lies in the range [0,1]. In semantic similarity, we look for defined relationships among various terms and concepts (e.g. defined in an ontology or extracted). In S^2R, both syntactic and semantic similarity filters are applied to find a set of Web services that match users' requirements. We assume that each Web service is defined using a description language such as WSDL (Web Service Description Language) which describes the functional service properties and its interface. WSDL files are published in a service registry that allows providers to advertise general information about their Web services. This information is used by clients for discovering providers and Web services of interest. UDDI and ebXML are examples of protocols that can be used for the registration of Web services. Since UDDI is the leading specification for the development of service-based repositories or registries [Bouguettaya et al. 2008] we use UDDI as a registration repository, where service providers publish their WSDL files (in catalog form). UDDI is organized in form of business activity categories (including the built-in NAICS, UN/SPSC and the other user defined categories), and service providers are responsible

for publishing their services in the appropriate UDDI category. Numerous Web services providing similar functionality may thus be listed under the same category in a UDDI.

In S^{2}R, we search the UDDI and retrieve the Web services under a category and send them to the first level (FCF). Thus, S^{2}R starts by calculating the syntactic similarity for each attribute, and if a syntactic match is not found, candidate services are checked for semantic similarity. The attributes types in FCF are: (1) syntactic attributes, which include the list of input and output parameters, the data types of the parameters, and the protocol to be used to invoke the Web service such as SOAP. (2) Semantic attributes, include the pre-conditions and effects of an operations execution.

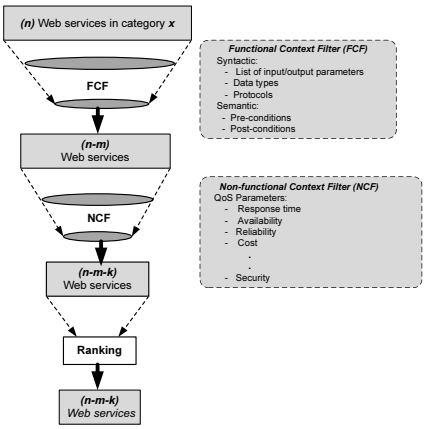


Figure. 2: Overview of the matching levels for S^{2}R.

We feed the (reduced) output set from (FCF) into the second level (NCF) filter (see Figure 2.). NCF is a filtering mechanism based on QoS parameters which measure the quality of a Web service. There are many parameters that can be used to measure a Web service's quality such as response time, availability, reliability, cost, security and privacy, etc. [Comuzzi and Pernici 2009] [Lee 2011] [Krishnamurthy and Babu 2012] [Pernici and Siadat 2011] [Yeom et al. 2011]. In addition, we can add any new specification to each one of these filters. After finding the providers that support the same service based on functional context (n-m providers), we filter them based on QoS requirements and are left with (n-m-k providers). In the third level, we rank the candidate Web services based on their utility. In the ranking level, we rank the candidate Web services based on their utility. We divide the Web services into two sets: HighRank set and LowRank set. The HighRank set includes the Web services that have QoS values higher or equal to consumer's requested values with the constraint that the price does not exceed consumer's maximum price. However, the LowRank set includes the Web services that have QoS values lower than the requested values. In case of empty HighRank set, the first Web service in the LowRank set is considered the best candidate. Note that the first two levels (i.e., FCF and NCF) are 'context based filters'.

A context is "any information that can be used to characterize the situation of an entity. An entity is any person, place, or subject that is considered relevant to interaction between a user and an application, including the user and the application themselves" [Dey 2000]. Context has been used in several areas such as machine learning, computer vision, information retrieval, and decision support [Kouadri Mostéfaoui and Brézillon 2006]. We view context as any Web service consumer or provider-related information that enables interactions between service consumers and providers. The provider-related context contains meta-data about the provider and its service (e.g., service description, QoS, etc). Similarly, consumer-related context contains meta-data about the consumer (e.g., consumer's location, expertise level, etc). For example, a non-functional

context policy may include a set of quality of service parameters (e.g., response time) associated with the service. Each context definition belongs to a certain category which can be either consumer-related or provider related. From a provider's perspective, interacting with a consumer depends on the situation (i.e. current variable values) of that consumer, and vice versa. Due to space restrictions, we omit further details regarding context definition. The interested reader is referred to [Medjahed and Atif 2007]. The summary of the levels and their parameters is given in Table I. In the following, we provide details for the S^{2}R filtering levels mentioned above.

Table I: S^{2}R Levels

Level	Context Filter	Context Type	Parameters	Supported language	References
First	FCF	Functional Context	Syntactic and semantic	OWL-S, WSDL-S, ..	[Martin et al. 2007], [Paolucci and Wagner 2006]
Second	NCF	Non-functional Context	response time, availability, reliability, cost,...	WSCL, HQML, ..	[Gu et al. 2007], [Gu et al. 2001]
Third	Ranking	Ranking	HighRank set and LowRank set	None	N/A

4.1 Level I: Functional Context Filter (FCF)

As mentioned earlier, the WSDL files are published in a UDDI and consist of (textual) descriptions of the Web service's operations (such as input, output, conditional output, precondition and postcondition). While some service providers describe these functionalities in different ways (e.g. both input and precondition are described as input), so S^{2}R includes preconditions with inputs and postconditions with outputs.

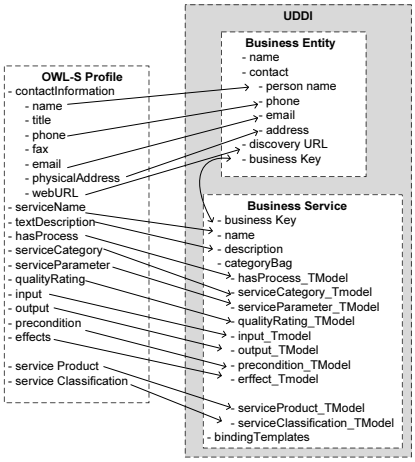


Figure. 3: Mapping between OWL-S and UDDI constructs.

In S^{2}R, we extract this and other functionalities' information from OWL-S. An OWL-S service is characterized by three types of knowledge:

(1) Service profile: it describes the operation of the service. It consists of three types of information: a human readable information section which describes the service, the functions that the service provides and a list of functional attributes. For example, hotel service provides the room availability of a specific hotel, this is the human information, the functional attribute is the input, the output and any other quality of service attribute such as the response time.

(2) Process-model: it describes how the service works by defining the services composition and the exact operations.
(3) Service grounding: it specifies the details of how an agent can access a service (i.e., the information needed by the agent to discover the service).

If the Web service does not support OWL-S, S^{2}R extracts the Web service information from the UDDI using the OWL-S/UDDI mapping as shown in Figure 3.

In an ideal scenario we would be able to find a service that perfectly matches to user requirements. However in SOAs with numerous combination of service attributes (i.e., input, output, and operations) the chances of having such a perfect match may be slim. Thus, instead of trying to find a perfect match, we could find a Web service that fulfills the user's requirements as much as possible (i.e., Web services may provide less functionalities or may have more functionalities than requested). In S^{2}R, we first look for a perfectly matching service, then we increase our search to incorporate services that provide more functionalities than requested. If we cannot find any suitable candidate in the first two searches, we expand our search to include services that provide less than desired functionalities. However, in such scenarios we would need to compose multiple services to provide the requested functionality. Thus, we may have the following four scenarios (see Figure 4).

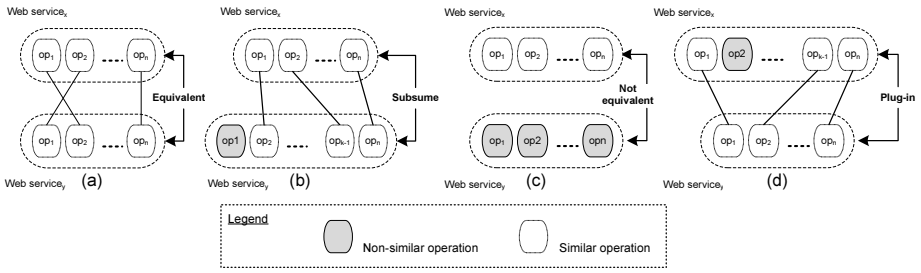


Figure. 4: Service matching scenarios.

• Equivalent (Figure 4a.) Web service$_x$ and Web service$_y$ are equivalent if all operations in Web service$_x$ are exactly the same as all operations in Web service$_y$ and the number of operations in Web service$_x$ is equal to the number of operations in Web service$_y$. Moreover, the inputs and outputs for each operation in Web service$_x$ are the same (names could potentially differ, e.g., cost vs price)as the inputs and outputs for each operation in Web service$_y$.

• Subsume (Figure 4b.) Web service$_x$ is subsumed by Web service$_y$ if all operations in Web service$_x$ are included in Web service$_y$. However, Web service$_y$ has extra inputs, outputs or operations. In this case Web service$_y$ can be counted as similar Web service to Web service$_x$ but it may request or provide extra information.

• Not-equivalent (Figure 4c.) Web service$_x$ and Web service$_y$ are not equivalent if all operations in Web service$_x$ do not match any operation in Web service$_y$. In this case Web service$_x$ and Web service$_y$ are totaly different.

• Plug-in (Figure 4d.) Web service$_y$ is plugged-in Web service$_x$ if some operations in Web service$_y$ matches some operations in Web service$_x$. In this case we need to find and compose another Web service(s) that cover the extra operations needed for Web service$_x$.

To classify any Web service under one of the matching scenarios in S^{2}R, we identify the service operations according to the following. We consider three main types of operations: one-way, request-response, and confirmation. In one-way operations, the Web service receives a message

without producing any output message (i.e., one-way communication). In request-response operation, the service receives an input message, processes it, and sends correlated output message to the sender. Confirmation operation sends an output message but does not expect to receive any more messages. `CW::Get quote::FW` (see Figure 1.) is an example of a request-response operation. Its input includes departure airport, arrival airport, departure date, return date, and the number of passengers. The output message for this request-response message contains a price and room type(s). `ChW::Notify consumer credit::CrW` (see Figure 1.) is a one way operation whose input contains a first name, last name, age and number of days. `TW::Confirmation` (see Figure 1.) is a confirmation operation with the output of reservation details and a receipt. As we can see Request-response operations have both input and output messages. However ,One way operations only contain input messages and confirmation operations only produce output messages. Each message consists of one or more parameters called parts in a WSDL. A parameter has a name and a data type. The data type gives the range of values that maybe assigned to the parameter. The first step in finding functional equivalence among Web services is to extract this parts information from the WSDL file for the parameters and return values of operations provided by candidate Web services.

Definition 3.1.1. Two operations op_{ik} and op_{jl} match if either (1) type of message for op_{ik} = "one-way" and type of message for op_{jl} = "confirmation"; or (2) type of message for op_{ik} = "request-response" and type of message for op_{jl} ="request-response".□

Definition 3.1.2. Each Web service is accessible via operations and each operation is identified by a tuple $< Description_{ij}, Mode_{ij}, Input_{ij}, Output_{ij}, Purpose_{ij}, Category_{ij}, Quality_{ij} >$, where $Description_{ij}$ is a textual summary about the features of the operation, $Mode_{ij}$ is the type of operation (i.e., one way, response-request or confirmation), $Input_{ij}$ is the input of the operation (if it exists), $Output_{ij}$ is the output of the operation (if it exists), $Purpose_{ij}$ is the business function offered by the operation, $Category_{ij}$ describes the operation domain, $Quality_{ij}$ provides the operation's qualitative properties. □

Example: The operation `TW::Get quote::HW` in our running example (Figure 1). is defined by a tuple <this operation returns the price for a given date to reserve hotel, request-response, dates and number of passengers; price in dollar, bussiness.function = request for quote, hotels, Quality.price>x and Quality.security="false">.

Definition 3.1.3. $operation_{ij}$ is similar to $operation_{kl}$ or subsumed by $operation_{ij}$ if

(1) $\forall x \in Input_{ij}, \exists x' \in Input_{kl} |$ x is data type compatible with x'.

(2) $\forall y \in Output_{ij}, \exists y' \in Output_{kl} |$ y is data type compatible with y'.

(3) $(Category_{ij} = Category_{kl}) \vee (Category_{ij} \subseteq Category_{kl})$.

(4) $Mode_{ij} = Mode_{kl}$ (see definition 3.1.1).

(5) $(Purpose_{ij} \equiv Purpose_{kl}) \vee (Purpose_{ij} \subseteq Purpose_{kl})$.

(6) Text matching between $Description_{ij}$ and $Description_{kl} \geqslant \ell | \ell$ is a pre-determined threshold.

In FCF, we apply neighborhood calculation to find the similar Web services based on their functional properties. This filter works based on three predefined matrices: input matrix, output matrix and operation matrix. Figure 5. (expanded version of Figure 2.) shows the steps of how FCF works. Upon arrival of a consumer request, a list of n services is retrieved from the UDDI under the requested category. FCF's Matrix Builder module then creates the three matrices: input, output and operation. Each matrix has $m \times n$ dimensions where m is the number of retrieved Web services and n is the number of inputs, outputs or operations for each matrix respectively. For instance, an $A_{m \times n}$ operations matrix is created as

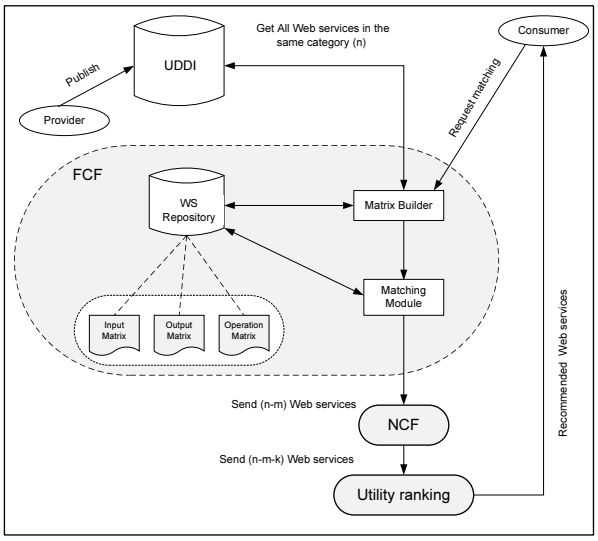


Figure. 5: Functional Context Filter (expanded).

$$A_{i,j} = \begin{cases} 1 & \text{if Web service}_i \text{ has operation}_j \ ; \\ 0 & \text{otherwise.} \end{cases}$$

When the Web service$_i$ does not provide the operation$_j$, the value of $A_{i,j}$ is zero. For example, if we have non-stop operation (i.e., provide the non-stop routes) and Web service$_x$ does not provide it, we will add zero under this operation for Web service$_x$.

Table II: Example of Web-operation Matrix

	Get-Quote	Get-Destination	Get-Price	Get-Time
Web service$_1$	1	1	0	1
Web service$_2$	1	1	1	0
Web service$_3$	0	1	1	0
Web service$_4$	1	1	1	1
Web service$_5$	1	1	0	0
Web service$_6$	0	1	1	1

While we are filling the matrix, the main concern is determining if the parameters of service$_x$ is the same as the parameters in the matrix. For instance, finding a flight using Web service$_x$ requires the input (airport name), but Web service$_y$ may requires the input (zip code) for the same operation. Hence it is important to find sematic similarity to address such scenarios.

S^{2}R extracts the semantic information of the candidate Web services through OWL-S. The semantics of the parameters are defined by the following attributes:

(1) Consumer and provider types: the consumer and the provider should be under the same category. For example, if they provide travel services then they should be under the travel category. In case of a composite solution that has multiple categories, the *Business role* will define the category for each service.

(2) Category: the category for each parameter describes the area of interest of the parameter. The category is defined by a tuple (domain, synonym, overlap). Domain gives the area of interest. For example, "travel" which takes these values from the domain in OWL-S. Synonym contains a set of alternative names for the domain name. For example, "trip" is a synonym of "travel". Overlap contains the list of categories that overlap with the current category.

(3) Purpose: describes the goal of the parameter, for example, the goal of `Get-Quote` in the scenario is to return the price of the requested service.
(4) Business role: the business role gives the type(category) information about a service under a certain business role. Every parameter has a well defined meaning according to the taxonomy.
(5) Unit: it is the measurement unit for a parameter such as, using miles to measure the distance and dollar to measure the cost, etc.

We use Table II to illustrate matrix building. The matrix contains six Web services and four operations. The matrix dimensions are $A_{6\times4}$. The operations for Web service$_1$ are (`Get-Quote`, `Get-Destination` and `Get-Time`), the Web service$_2$ operations are (`Get-Quote`, `Get-Destination` and `Get-Price`), etc. The first step of S^{2}R is determining the inputs, outputs and operations of the Web service which based on the consumer request. If all the properties are available in the matrix then we just add the service name, and insert one under the property if the service provides it, else insert zero. However, if the property does not exist, we will edit and add the new property to the matrix. For example, one provider wants to publish Web service$_7$ which includes the operations (`Get-Quote`, `Get-Destination`, `Get-Price` and `Get-Rate`). The first three operations exist in the matrix but the last operation is a new one. In this case we add the new property (`Get-Rate`), add one under the operations (`Get-Quote`, `Get-Destination` and `Get-Price`), so the new matrix will be as follows

Table III: Example of Adding Web service to the Web-operation Matrix

	Get-Quote	Get-Destination	Get-Price	Get-Time	Get-Rate
Web service$_1$	1	1	0	1	0
Web service$_2$	1	1	1	0	0
Web service$_3$	0	1	1	0	0
Web service$_4$	1	1	1	1	0
Web service$_5$	1	1	0	0	0
Web service$_6$	0	1	1	1	0
Web service$_7$	0	1	1	1	1

FCF inserts the requirements into a vector by getting the parameters for a specific category from the service repository. It then builds a priority matrix. The priority matrix is a matrix that gives weight to each property and will move the focus towards more important operations. Based on TF-IDF [Karimzadehgan et al. 2011], we define the priority matrix over the original matrix $A_{m\times n}$ to compute the weight of each item as:

$$w_{i,j} = \frac{A_{i,j} \times |Ws_i|}{OpM} * log\frac{A_{i,j}}{|Op_j|} \tag{1}$$

where $A_{i,j}$ is one if the operation j exists in Web service i, otherwise $A_{i,j}$ is zero, $|Op_j|$ is the number of times that Op_j has been used by all Web services, OpM is the number of operations in the matrix and $|Ws_i|$ is the number of operations for Web service i. The result after applying Equation 1. to our example matrix (in Table III) will be the priority matrix in Table IV. The operations that are provided most often by the Web services in the same category will have the highest weights and the operations that are provided less will have lower weights.

After building the priority matrix, FCF converts each row of the matrix into binary vectors, for example if the consumer request contains the operations <`Get-Quote`, `Get-Destination`, `Get-Price`, `Get-Rate`> while the available service has <`Get-Quote`, `Get-Destination`, `Get-Price`, `Get-Time`, `Get-Rate` > then the query vector of this Web service is $< 1, 1, 1, 0, 1 >$. FCF finds similar Web services based on vector similarity [Chan et al. 2011] as:

$$Similarity(I, J) = |cosine\overrightarrow{V_i}, \overrightarrow{V_j}| = |\sum_{k=1}^{n}(i_k \times j_k)| \div \sqrt{\sum_{k=1}^{n} i_k^2} \times \sqrt{\sum_{k=1}^{n} j_k^2} \tag{2}$$

Table IV: Priority Matrix

	Get-Quote	Get-Destination	Get-Price	Get-Time	Get-Rate
Web service$_1$	0.0635	0.1111	0	0.0635	0
Web service$_2$	0.0635	0.1111	0.0794	0	0
Web service$_3$	0	0.1667	0.1190	0	0
Web service$_4$	0.0476	0.0833	0.0595	0.0476	0
Web service$_5$	0.0953	0.1667	0	0	0
Web service$_6$	0	0.1111	0.0794	0.0635	0
Web service$_7$	0	0.0833	0.0595	0.0476	0.0119

Now, let us suppose that the consumer requests a matching for Web service that includes the operations: $< \texttt{Get}-\texttt{Quote}, \texttt{Get}-\texttt{Destination}, \texttt{Get}-\texttt{Price} >$ i.e., the vector will be $\overrightarrow{V_1} =< 1,1,1,0,0 >$ and it will compared to all vectors $\overrightarrow{V_d}$ where d $\in$ [1,m] in the matrix $A_{m\times n}$.

$$I = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}, \qquad J = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \end{pmatrix}, \qquad Similarity(I,J) = \begin{pmatrix} 0.6667 \\ 1.0 \\ 0.8403 \\ 0.8824 \\ 0.8403 \\ 0.6667 \\ 0.5882 \end{pmatrix} \tag{3}$$

If the threshold β for selecting the Web services is (0.8) then the result from FCF is the set $\{Webservice_2, Webservice_3, Webservice_4, Webservice_5\}$. In this case, we find that $Webservice_3$ and $Webservice_5$ have the same similarity value (0.8403). Notice that $Webservice_3$ does not provide the operation `Get-Quote` and $Webservice_5$ does not provide the operation `Get-Price`. In such case, we return back to the operation priority matrix which shows the priority for the operation `Get-Price` is 0.1190 and the priority for the operation `Get-Quote` is 0.0953, so we prefer $Webservice_3$ contains the higher priority operation.

4.2 Level II: Non-functional Context Filter (NCF)

NCF is divided into two steps: The first step checks for the service *availability*, thereby, eliminating the Web services that are unavailable. The second step checks Web service similarity based on other QoS parameters (e.g., response time, throughput, reliability, etc). We use the ping utility for the former, which has been used for Web service performance measurements [Guoping et al. 2009]. After determining the set of Web services that respond to the ping inquires, we start the second step where we use Context Policy Assistance (CPA) to test the similarity between the QoS parameters that are required by the consumer and the QoS parameters offered by the available Web services.

```
Context Rule  NF-cost
    Context Property  Cost
    Instance  cost_s, cost_d
    Type  NFP
    Action  matchproperity($cost_s$, $cost_d$)
      { If cost_d<= cost_s Then return true else return false }
```

Figure. 6: Rule example.

CPA is created by the service provider and should be attached to the service. It facilitates interaction between providers and the service registry to store the context policies. For further details, the interested reader is referred to [Medjahed and Atif 2007]. A service provider may create the context specification using a context specification language such as WS-Policy. WS-Policy provides a general model and syntax to describe and communicate the policies of Web services [Erradi et al. 2007]. Each policy contains a set of rules that define the QoS requirements/capabilities of the Web services. A sample rule is shown in Figure 6. where a context

rule is identified by a name to specify the property. In this rule we need two instances: the consumer's cost and the provider's cost. The type of this policy is NFP (non-functional policy) and it compares the cost between the two parties to determine if they are compatible. We use these policies to determine if two Web services are similar based on the QoS parameters they both share [Alrifai et al. 2010].

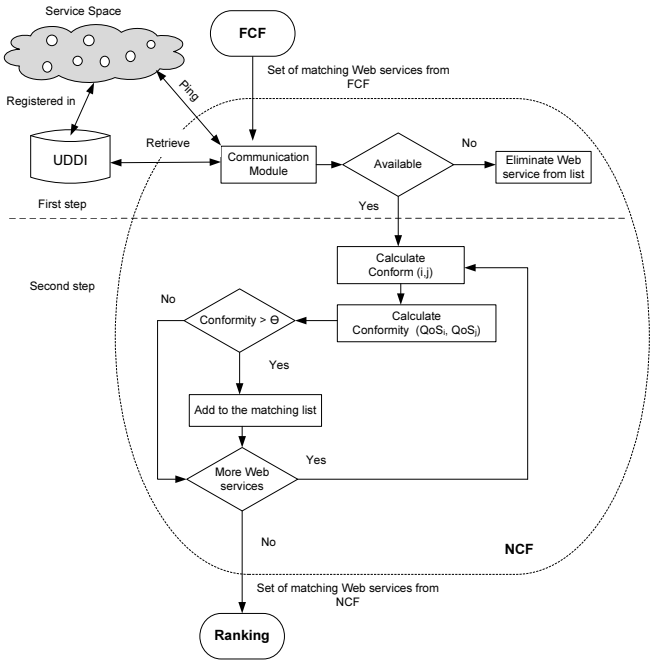


Figure. 7: Non-functional Filter (expanded).

Definition 3.2.1. QoS vector description: it is an extendable vector used to define QoS parameters of the provider and the consumer. It is expressed as: QoS = $< QoS_1, QoS_2, ..., QoS_n >$, n ∈ R, where QoS_n indicates n^{th} QoS attributes and QoS_i where i ∈ [1,n] is equal to {Availability, Cost, Response time, Error rate, Throughput, Reliability, Reputation, and Security.}.□
QoS parameter matching is done as:

(1) Convert the parameters into QoS vector descriptions. Then, we have one vector for the consumer request $< ConQoS_1, ConQoS_2, ..., ConQoS_n >$ and multiple vectors for the provider offerings $< Pro_1QoS_1, Pro_1QoS_2, ..., Pro_1QoS_n > ... < Pro_kQoS_1, Prov_kQoS_2, ..., Prov_kQoS_n >$. Here we assume that the providers provide trusted information for the QoS values [Sherchan et al. 2010]. There are three different cases in converting the QoS parameters: the consumer vector is greater than the provider vector, the consumer vector is less than the provider vector, or the consumer vector is equal to the provider vector. In the first case we add zeros at the end of the consumer vector and in the second case we add zeros at the end of provider vector.

(2) Create a new vector called conform with length equal to the max length of consumer and provider vectors. For each element in the vector use the polices to compare the conditions, and if the condition is met then add one to the conform vector else put zero. At the end of this step we will have the vector $< conform_1, conform_2, ..., conform_n >$.

(3) Calculate the conformity degree between the services for the consumer QoS_i and the provider QoS_j as:

$$Conformity(QoS_i, QoS_j) = \sum_{q=1}^{z} Weight_q * conform_q \quad (4)$$

where i,j are Web services, z is the maximum length of parameters, i.e., $z = max(|QoS_i|, |QoS_j|)$, and $Weight_q$ is the weight assigned to each QoS parameter.

Consumers may have different expectations about the conformity degree of their services. For this purpose, they provide a conformity threshold θ ($0 < \theta \leqslant 1$). In NFC, we find all Web services j where the conformity degree (QoS_i, QoS_j) is greater than θ, which are then passed to the ranking level. The conformity threshold is given by the consumer as a part of his profile, while the QoS weight is created automatically by the system based on the level of the consumer's expertise. In S^{2}R, we defined three types of consumers: expert, regular and normal consumers. The expert consumers are knowledgable about meaning of all QoS parameters and they may assign the desired value for the QoS parameters for specific services. The regular consumers have some knowledge about the QoS parameters, and they may assign values to some QoS parameters and leave the other parameters without weights. In this case, the system predefined weights are used for unassigned parameters. The normal consumers do not have any knowledge about the QoS parameters that the system assigns weights for all parameters.

The main benefit of categorizing the consumers into these types is to let the expert consumer participate in making a decision by providing weights for each QoS parameter. However, the system will provide all the QoS attribute weights for other categories of the consumers. In essence, consumer categories are determined based on the assigned values for the current request.

Example: Suppose that Web service HW (see Figure 1) is one of the candidate Web services as a result of FCF. Let it have the following QoS vector: < price = \$50; response-time= 60 sec; error-rate = 0.01; security="false">. On the other hand, the consumer (Web service$_x$) QoS vector is as following: < price $\leqslant$ \$70; response-time < 90 sec; error-rate < 0.05; reliability= 0.80; security="true">. The first step is building the conformity vector: $< 1, 1, 1, 0, 0 >$. The first three values of the vector are equal to one because the conditions are met between Web service$_x$ and Web service$_y$. However, since the values of reliability and security do not match, 0's are appended. The conformity degree based on individual QoS parameter weights is then assessed. Assume that the consumer provides the weights as $< 0.3, 0.2, 0.4, 0.1, 0.3 >$, then:

$$Conformity(QoS_x, QoS_y) = \sum_{q=1}^{5} < 0.3, 0.4, 0.1, 0.1, 0.1 > * < 1, 1, 1, 0, 0 >= 0.8 \quad (5)$$

4.3 Level III: Web service Ranking

In this level, S^{2}R ranks the Web services based on the range compatibility of the QoS parameters. We use weighted sum filter function after converting the QoS parameters into a range vector in the format of the component vector description.

Definition 3.3.1. The component vector description is expressed as:
$< (QoS1, QoS1_{min}, QoS1_{max}), (QoS2, QoS2_{min}, QoS2_{max}), ..., (QoSn, QoSn_{min}, QoSn_{max}) >$,
n $\in$ R, where QoSi$_{\in[1,n]}$ is the best QoS value for parameter i, $QoSi_{min}$ is the minimum acceptable value for parameter i, and $QoSi_{max}$ is the maximum acceptable value for parameter i, then $QoSi_{min} \leqslant QoSi \leqslant QoSi_{max}$.□

Each vector is accompanied by a decision model, i.e. ranges of all the QoS parameters as well as their respective priorities also known as the weights. The ranking will be based on the matching degree (i.e., how near the QoS parameter that is provided by the provider is to the

QoS parameters required by the consumer). Note that we can use the provider QoS values in one of two ways:(i) QoS values as advertised by the service provider, or (ii) QoS values obtained using behavior monitoring through the community (i.e., provider reputation). The best ranked Web service will be the Web service that is closest to the 'best' value and the worst ranked will be the Web service that is the farthest from the 'best' value. However, if a Web service provides a larger value than the best value but has a lower cost associated to it, then the system will give this Web service a higher rank.

Definition 3.3.2. $\forall$ Web services $WS_i \in \omega$, where ω is the set of all Web services which are similar (functional and non-functional) to the requested service:

$$WebServiceRankSet = \begin{cases} HighRank & \text{if } WS_i \geqslant \mu \wedge cost \leqslant \wp; \\ LowRank & \text{if } WS_i < \mu. \end{cases}$$

where μ is the best value of QoS_j that is provided by the consumer from the vector $< (QoS_j, QoS_{jmin}, QoS_{jmax}) >$, and $\wp$ is the acceptable cost by the consumer. □

We assume that all the participating Web services are able to articulate their objectives and prioritize them [Ackoff 1978]. The articulation and prioritization of objective values is well accepted in multi-attribute situations and operations research [Chandra et al. 2000] [Faratin et al. 2002] [Resinas et al. 2012]. The consumer determines/assigns a priority for each QoS parameter (e.g., the price of the service is more important than its execution time). In our method, we covert these priorities into a weighted vector to compare the consumer requirements with the provider offer. Table V lists the definition of symbols used henceforth. All the Web services conform to some constraints in the solution. For instance, any QoS vector cannot have a negative value (as shown by Equation 6), and the QoS values lie between the maximum and minimum allowable values set by the consumer Web service (as shown by Equation 7).

$$X_j \geqslant 0 \quad and \quad Y_{ij} \geqslant 0 \tag{6}$$

$$X_{j(min)} \leqslant X_j \leqslant X_{j(max)} \quad and \quad Y_{ij(min)} \leqslant Y_{ij} \leqslant Y_{ij(max)} \tag{7}$$

The utility function is a multi-step calculation that evaluates the degree of matching between the Web services. A weighted sum approach is used to combine these multiple QoS parameters. We use a distance function to measure the difference among the proposed solutions of both the consumer and provider Web services. Thus, lower utility values are desired as they translate to lesser mismatch among the services. Similarly, lower values translate to higher ranks for the solutions among the solution space. The utility value of a match is calculated as follows

$$\Delta_{ij} = \frac{|X_j - Y_{ij}|}{X_j} \tag{8}$$

$$r_j = \sum_{j=0}^{n} (WX_j * \Delta_{ij} + WY_{ij} * \Delta_{ij}) \tag{9}$$

$$R_s = min \sum_{j=0}^{G} (r_j) \tag{10}$$

Based on the previous equations (Equation 8, Equation 9, and Equation 10) we compare the maximum QoS for each Web service in the set. If the maximum QoS is greater than or equal R_s then this Web service will be in the ranking set. At the end of this level we have an ordered list of Web services based on their QoS parameters. Below is the algorithm of our technique which contains two subfunctions FCF and NCF shown in Figure 8.

Algorithm 1 Semantic Web services Matching Algorithm

INPUT: A Web service IP address.
OUTPUT: A set of ranked similar Web services..
Contact UDDI to determine the set of Web services (n) under the same category of Web service$_x$.
Retrieve the inputs, the outputs and the operations for (n) Web services.
Υ = **Call** FCF(n, x.paramaters, j.parameters)
λ = **Call** NFC(Υ, consumer's experience, $< QoSx_1, QoSx_2, ..., QoSx_q >$)
$V_1 =< QoSx_i, QoSx_i(min), QoSx_i(max) >$
$V_2 =< QoSj_i, QoSj_i(min), QoSj_i(max) >$
Set ω=$\emptyset$
for each Web service T $\in \lambda$ **do**
  $X_j \geqslant 0$ and $Y_{ij} \geqslant 0$
  $X_{j(min)} \leqslant X_j \leqslant X_{j(max)}$ and $Y_{ij(min)} \leqslant Y_{ij} \leqslant Y_{ij(max)}$
  $\Delta_{ij} = \frac{|X_j - Y_{ij}|}{X_j}$
  $r_j = \sum_{j=0}^{n} (WX_j * \Delta_{ij} + WY_{ij} * \Delta_{ij})$
  $R_s = min \sum_{j=0}^{G} (r_j)$
  if QoS$_T$(max)$\leqslant R_s$ **then**
    T $\in \omega$
  end if
end for
return ω

Input: A set of WS$_j$(j $\in$[1,n]),
x.input$_i$, x.output$_i$,x.operation$_i$,
j.input$_i$, j.output$_i$,j.operation$_i$,
Output: A set of functional similar Web services(Υ).
for each Web service $s_j \in n$ **do**
  for each x.parameter$_i$ **do**
    if s_j.parameter$_k \equiv$ x.parameter$_i$ **then**
      A(i,j)=1
    else {s_j.parameter$_i \neq$ x.parameter$_i$}
      A(i,j)=0
    end if
  end for
end for
for i =1 to m **do**
  for j=1 to n **do**
    $w_{i,j} = \frac{A_{i,j} \times |Ws_i|}{|Op|} * log \frac{A_{i,j}}{|Op_j|}$
  end for
end for
$Similarity(x,y) = |cosine\overrightarrow{V_i}, \overrightarrow{V_j}| = |\sum_{k=1}^{n}(i_k \times j_k)| \div \sqrt{\sum_{k=1}^{n} i_k^2} \times \sqrt{\sum_{k=1}^{n} j_k^2}$
Set Υ=$\emptyset$
for each Web service$_y$ **do**
  if Similarity(x,y)¿β **then**
    Web service$_y \in \Upsilon$
  end if
end for
return Υ

(a) Functional context Filter(FCF)

Input: A set of Web services (Υ), consumer's experience, QoS vector for Web service$_x$ and WS $\in \Upsilon$
Output: A set of non-functional similar Web services(λ).
Set λ=$\emptyset$ and $\Re$=$\emptyset$
for each Web service R $\in \Upsilon$ **do**
  if Ping(R.IP) is TRUE **then**
    R $\in \Re$
  end if
end for
for each T $\in \Re$ **do**
  for i=1 to n **do**
    for j=1 to m **do**
      if $QOSx_i \equiv QoSR_j$ **then**
        $Conform_i$ =1
      else
        $Conform_i$ =0
      end if
    end for
  end for
  $Conformity(QoSx, QoSR) = \sum_{q=1}^{z} Weight_q * conform_q$
  if Conformity (QoSx,QoSR)$\geqslant \theta$ **then**
    R $\in \lambda$
  end if
end for
return λ

(b) Non-functional context Filter(NCF)

Figure. 8: Functional and Non-functional Context Filters

5. PERFORMANCE ANALYSIS

In this section, we define an analytical model to study the performance of the proposed technique (S^2R). Our Analytical model has 1000 Web services that are divided into three categories. We change the number of polices that are evaluated every time (e.g., 2, 4, 8 policies) while keeping other variables such as number of context specifications per policy, number of members per category, etc. fixed. We focus on computing the total time and search space complexity for checking the similarity degree of the target Web services through our three levels. We compare our technique with three similar existing works through this analytical model.

Table V: Definition of Symbols

Symbol	Definition
X_j	The value of jth component of consumer's vector.
Y_{ij}	The value of jth component of ith Provider's vector.
$X_{j(min)}$	The minimum allowed value of jth component of consumer's vector as provided by the consumer.
$X_{j(max)}$	The maximum allowed value of jth component of consumer's vector as provided by the consumer.
$Y_{ij(min)}$	The minimum allowed value of jth component of ith Provider's vector as provided by the provider.
$Y_{ij(max)}$	The maximum allowed value of jth component of ith Provider's vector as provided by the provider.
WX_j	The weight of jth component of consumer's vector as provided by the consumer.
WY_{ij}	The weight of jth component of ith Provider's vector as provided by the provider.
r_j	Utility of the solution s for participant j.
R_s	Utility of the solution s (for all participants).
N_{com}	Number of categories.
N_p	Number of policies per service.
N_{cs}	Number of context specifications per policy.
N_{member}	Number of members per category.
N_r	Number of rules.
N_s	Number of services in a particular category.
T_{pall}	Time to fetch all policies of a service.
T_{pall_1}	The time to parse a service description and the network transmission delay.
T_{pall_2}	The time spent by each category to process its sub request.
T_{pone}	Time to fetch one policy of a service.
T_{cs}	Time to get a context specification.
T_{XML}	Time to parse a service description.
T_{Net}	Network transmission delay.
T_{rep}	Time spent to assess a reply from a category.

Table V defines the parameters and symbols used here after. We assume that Web services are divided into categories (e.g., under UDDI categories). To simplify the analysis, we assume that the times to retrieve a description from a service registry and parse that description are fixed values. Based on the categories, we compute the average matching time T_{match} which is the time it takes to find the similar services.

$$T_{match} = \frac{(T_{MINmatch} + T_{MAXmatch})}{2} \tag{11}$$

where $T_{MINmatch}$ is the best case matching time and $T_{MAXmatch}$ is the worst case matching time. T_{match} includes a polling time (T_{poll}) and a decision time (T_{dec}), where, T_{poll} is a combination of T_{poll1} and T_{poll2}. T_{poll1} includes the time it takes to fetch the policies, parse a service description and the network transmission delay. In the best case, the Web service would have only a single policy ($T_{MINpoll1}$) and in the worst case it may have N_{com} polices ($T_{MAXpoll1}$). Hence,

$$T_{MINpoll1} = T_{pone} + T_{XML} + T_{Net} \tag{12}$$

$$T_{MAXpoll1} = T_{pone} + N_{com} \times (T_{XML} + T_{Net}) \tag{13}$$

T_{poll2} includes the time spent in each category to process its sub request.

$$T_{MINpoll2} = 2 \times T_{pone} + 2 \times T_{cs} + 4 \times T_{XML} \quad (14)$$

We multiply T_{cs} by two because we need to compare each policy twice: once for the provider and once for the consumer. At a minimum, each policy would be compared to a single policy on both sides. Similarly, we multiply T_{XML} by four because we need to parse the description of XML four times (we need to parse XML files twice for the consumer and twice for the provider: once for determining the properties and once for determining the policies).

$$T_{MAXpoll2} = T_{pone} + N_{cs} \times (T_{XML} + N_s \times (T_{pone} + T_{XML} + 2 \times Tcs + N_r \times (2 \times T_{XML}))) \quad (15)$$

In calculating $T_{MINpoll2}$ we multiply T_{pone} by two because we retrieve the policy for the source and the category member. The decision time (T_{dec}) includes the network delay and the time spent to asses a reply from the Web services under the same category. In the best case,

$$T_{MINdec} = T_{Net} + T_{rep} \quad (16)$$

$$T_{MAXdec} = N_{com} \times (T_{Net} + T_{rep}) \quad (17)$$

Based on the previous equations, T_{match} is then,

$$T_{match} = \frac{(T_{MINpoll1} + T_{MAXpoll1} + T_{MINpoll2} + N_{com} \times T_{MAXpoll2} + T_{MINdec} + T_{MAXdec})}{2} \quad (18)$$

The previous formulas give matching times for each technique. In what follows, we calculate the total matching time for the four techniques: S^{2}R, CME, CCB and Brute-force. Figure 9a. shows a comparison between the Brute-force method (exhaustive search), CME [Medjahed and Atif 2007], CCB [Segev 2008] and S^{2}R for service matching time based on the number of services. Note that Brute-force has the highest matching time especially when we have a large number of services. CME perfroms better than Brute-force method, but still takes more time than CCB. However, our method provides the lowest matching time, even if the number of services is large. Figure 9b. shows the relationship between the number of services and the search space for each matching method. We can see that Brute-force method has the largest search space and the smallest search is attributed to S^{2}R.

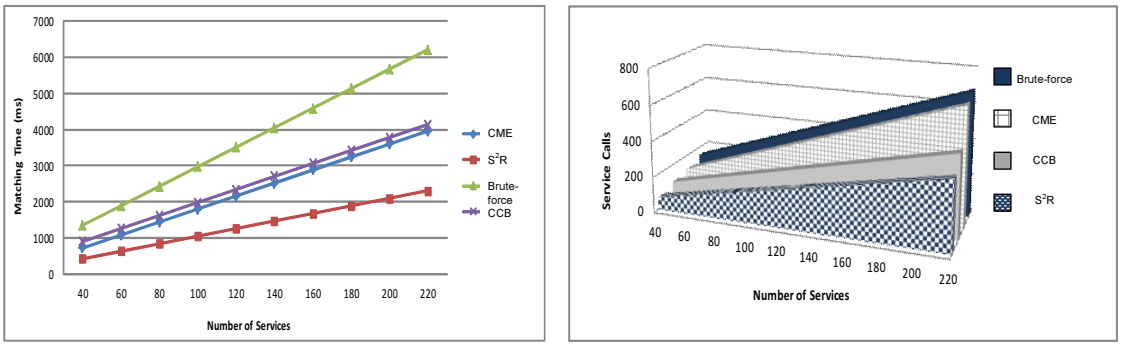


Figure. 9: Matching Time and Search Space analysis.

In the last set of experiments, we evaluate the services matching time with variable number of policies. Figure 10 presents the results of two, four, and eight policies. In Figure 10a. we can see that S^{2}R took a maximum matching time of (8,000 and 10,000 ms) when we use eight policies. However, in the Brute-force method (see Figure 10c.) the maximum time is between (22,000 and

25,000 ms). Figure 10b. shows that the maximum time using CME which is between (14,500 and 16,000 ms). Moreover, Figure 10d. shows that CCB took the maximum time of (15,000 to 17,000 ms). This shows that S^{2}R provides better matching time for all variable number of policies.

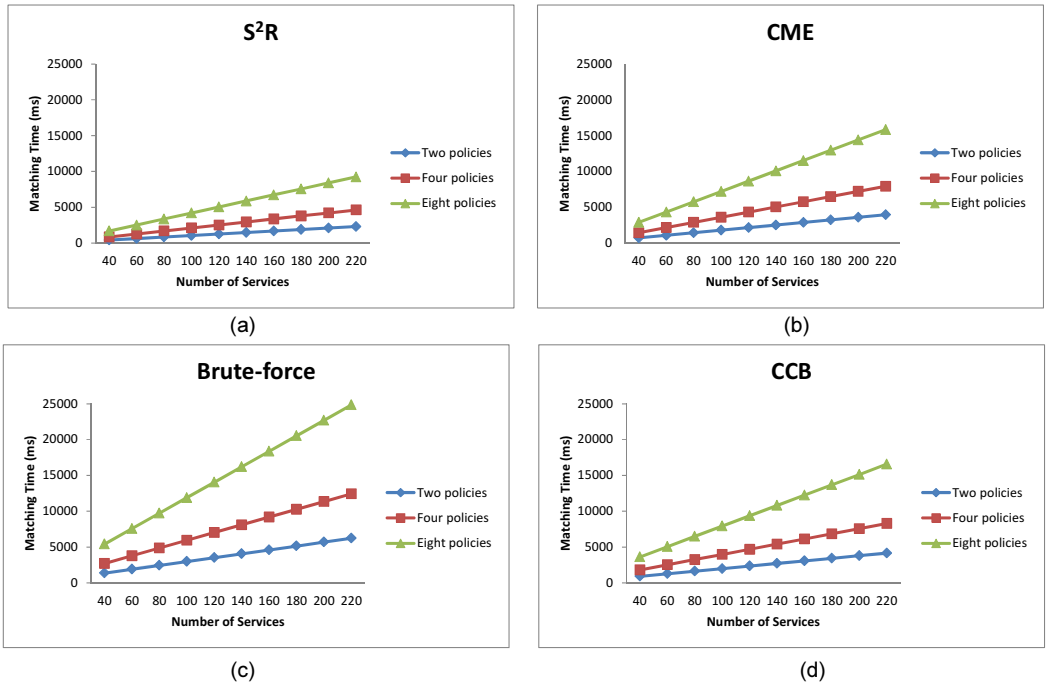


Figure. 10: Scalability analysis for different number of context policies. (a) S^{2}R. (b) CME. (c) Brute-force. (d) CBB.

Figure 11 shows the four techniques with their maximum and minimum matching times. We can see that S^{2}R takes the least amount of time to find the matching services, and scales very well when the number of Web services is increased (shown in Figure 11 from 40 to 220).

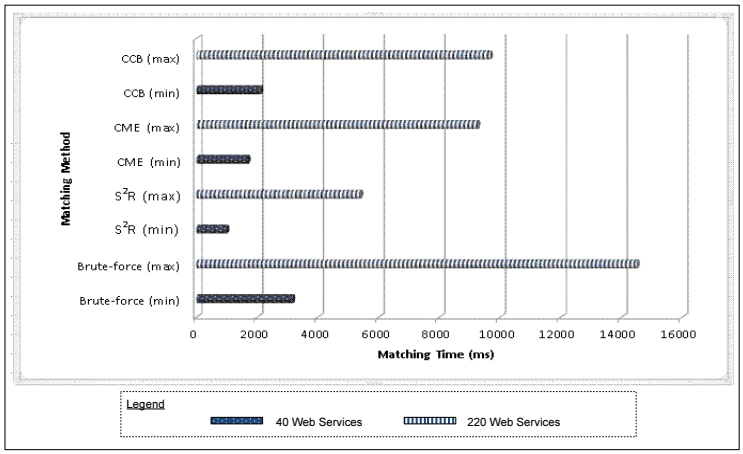


Figure. 11: Maximum and minimum matching times.

6. CONCLUSION

S^{2}R extends the scope of Web service selection by providing a semantic Web service similarity and ranking approach. Rather than selecting Web services randomly, we provide a selection of services that have the highest number of operations in coherence with the consumer request, and have corresponding *acceptable* QoS parameter values. Our method combines semantic and syntactic matching with QoS requirements. In addition, our method ranks the available candidate services to provide the user with a list of candidate services even if no exact match is found. Experiment results show that our proposed technique improves the service selection process by reducing the time and search space complexity. In the future, we plan to investigate techniques for enabling automated planning and replacement of faulty Web services with similar ones that have high utility.

REFERENCES

Ackoff, R. L. 1978. *Redesigning the future*. Wiley, New York.

Alhosban, A., Hashmi, K., Malik, Z., and Medjahed, B. 2011. Assessing fault occurrence likelihood for service-oriented systems. In *Proceedings of the 11th International Conference on Web Engineering*. 59–73.

Alrifai, M., Skoutas, D., and Risse, T. 2010. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th international conference*. WWW '10. ACM, New York, USA, 11–20.

Baïna, K., Benali, K., and Godart, C. 2001. A process service model for dynamic enterprise process interconnection. In *Proceedings of the 9th International Conference on Cooperative Information Systems*. CooplS '01. Springer-Verlag, London, UK, 239–254.

Bergmann, R., Richter, M. M., Schmitt, S., Stahl, A., and Vollrath, I. 2001. Utility-oriented matching: A new research direction for case-based reasoning. In *In professionlles wissens managment: Erfahrungen Und Visionen. Proceeding of the 1st conference knowledgr management. Shaker*. 264–274.

Bouguettaya, A., Krüger, I., and Margaria, T., Eds. 2008. *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*. Lecture Notes in Computer Science, vol. 5364.

Chakraborty, D., Perich, F., Joshi, A., Finin, T. W., and Yesha, Y. 2002. A reactive service composition architecture for pervasive computing environments. In *Proceedings of the IFIP TC6/WG6.8 Working Conference on Personal Wireless Communications*. PWC '02. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 53–62.

Chan, N. N., Gaaloul, W., and Tata, S. 2011. A web service recommender system using vector space model and latent semantic indexing. *Advanced Information Networking and Applications, International Conference on 0*, 602–609.

Chandra, S., Ellis, C. S., and Vahdat, A. 2000. Differentiated multimedia web services using quality aware transcoding. In *INFOCOM*. 961–969.

Comuzzi, M. and Pernici, B. 2009. A framework for qos-based web service contracting. *ACM Trans. Web 3*, 10:1–10:52.

Dey, A. K. 2000. Providing architectural support for building context-aware applications. Ph.D. thesis, Atlanta, GA, USA. AAI9994400.

Erradi, A., Maheshwari, P., and Tosic, V. 2007. Ws-policy based monitoring of composite web services. In *Proceedings of the Fifth European Conference on Web Services*. IEEE Computer Society, Washington, DC, USA, 99–108.

Faratin, P., Sierra, C., and Jennings, N. R. 2002. Using similarity criteria to make issue trade-offs in automated negotiations. *Artif. Intell. 142,* 2, 205–237.

Gu, X., Nahrstedt, K., Yuan, W., Wichadakul, D., and Xu, D. 2001. An xml-based quality of service enabling language for the web. Tech. rep., Champaign, IL, USA.

Gu, Z., Li, J., Tang, J., Xu, B., and Huang, R. 2007. Verification of web service conversations specified in wscl. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02*. COMPSAC '07. IEEE Computer Society, Washington, DC, USA, 432–437.

Guoping, Z., Huijuan, Z., and Zhibin, W. 2009. A qos-based web services selection method for dynamic web service composition. In *Proceedings of the 2009 First International Workshop on Education Technology and Computer Science - Volume 03*. ETCS '09. IEEE Computer Society, Washington, DC, USA, 832–835.

Heuvel, W.-J. v. d., Yang, J., and Papazoglou, M. P. 2001. Service representation, discovery, and composition for e-marketplaces. In *Proceedings of the 9th International Conference on Cooperative Information Systems*. CooplS '01. Springer-Verlag, London, UK, 270–284.

Karimzadehgan, M., Li, W., Zhang, R., and Mao, J. 2011. A stochastic learning-to-rank algorithm and its application to contextual advertising. In *Proceedings of the 20th international conference on World wide web*. WWW '11. ACM, New York, NY, USA, 377–386.

Kouadri Mostéfaoui, G. and Brézillon, P. 2006. Context-based constraints in security: Motivations and first approach. *Electron. Notes Theor. Comput. Sci. 146*, 85–100.

Krishnamurthy, V. and Babu, C. 2012. Pattern based adaptation for service oriented applications. *SIGSOFT Softw. Eng. Notes 37,* 1 (Jan.), 1–6.

Lee, C. and Helal, S. 2003. Context attributes: An approach to enable context-awareness for service discovery. In *Proceedings of the 2003 Symposium on Applications and the Internet*. SAINT '03. IEEE Computer Society, Washington, DC, USA, 22–.

Lee, Y. 2011. bqos(business qos) parameters for soa quality rating. In *FGIT-ASEA/DRBC/EL*. 497–504.

Li, B., Xu, Y., Wu, J., and Zhu, J. 2012. A petri-net and qos based model for automatic web service composition. *JSW 7,* 1, 149–155.

Li, L. and Horrocks, I. 2003. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th international conference*. WWW '03. ACM, New York, USA, 331–339.

Maamar, Z., Benslimane, D., and Narendra, N. C. 2006. What can context do for web services? *Commun. ACM 49*, 98–103.

Martin, D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., Sycara, K., Mcguinness, D. L., Sirin, E., and Srinivasan, N. 2007. Bringing semantics to web services with owl-s. *World Wide Web 10*, 243–277.

Mecella, M., Pernici, B., and Craca, P. 2001. Compatibility of e -services in a cooperative multi-platform environment. In *Proceedings of the Second International Workshop on Technologies for E-Services*. TES '01. Springer-Verlag, London, UK, 44–57.

Medjahed, B. and Atif, Y. 2007. Context-based matching for web service composition. *Distrib. Parallel Databases 21*, 5–37.

Menascé, D. A. and Dubey, V. K. 2007. Utility-based qos brokering in service oriented architectures. In *ICWS*. 422–430.

Narayanan, S. and McIlraith, S. A. 2002. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference*. WWW '02. ACM, New York, USA, 77–88.

Nepal, S., Sherchan, W., Hunklinger, J., and Bouguettaya, A. 2010. A fuzzy trust management framework for service web. In *ICWS*. 321–328.

Paolucci, M. and Wagner, M. 2006. Grounding owl-s in wsdl-s. In *Proceedings of the IEEE International Conference on Web Services*. IEEE Computer Society, Washington, DC, USA, 913–914.

Papazoglou, M. P., Pohl, K., Parkin, M., and Metzger, A., Eds. 2010. *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*. Lecture Notes in Computer Science, vol. 6500. Springer.

Pernici, B. and Siadat, S. H. 2011. Adaptation of web services based on qos satisfaction. In *Proceedings of the 2010 international conference*. ICSOC'10. Springer-Verlag, Berlin, Heidelberg, 65–75.

Resinas, M., Fernandez, P., and Corchuelo, R. 2012. A bargaining-specific architecture for supporting automated service agreement negotiation systems. *Sci. Comput. Program. 77,* 1, 4–28.

Segev, A. 2008. Circular context-based semantic matching to identify web service composition. In *Proceedings of the 2008 international workshop on Context enabled source and service selection, integration and adaptation: organized with the 17th International World Wide Web Conference (WWW 2008)*. CSSSIA '08. ACM, New York, NY, USA, 7:1–7:5.

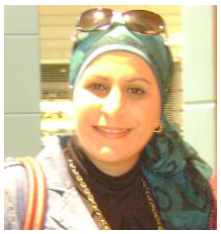
Sherchan, W., Nepal, S., Hunklinger, J., and Bouguettaya, A. 2010. A trust ontology for semantic services. In *IEEE SCC*. 313–320.

Sycara, K., Klusch, M., Widoff, S., and Lu, J. 1999. Dynamic service matchmaking among agents in open information environments. *SIGMOD Rec. 28*, 47–53.

Xia, H. and Yoshida, T. 2007. Web service recommendation with ontology-based similarity measure. In *Proceedings of the Second International Conference on Innovative Computing, Informatio and Control*. ICICIC '07. IEEE Computer Society, Washington, DC, USA, 412–.

Yao, D., Lu, B., Fu, F., and Ji, Y. 2010. A risk assessment algorithm based on utility theory. In *Proceedings of the Advanced intelligent computing theories and applications, and 6th international conference on Intelligent computing*. ICIC'10. Springer-Verlag, Berlin, Heidelberg, 572–579.

Yeom, G., Tsai, W.-T., Bai, X., and Lee, Y. 2011. A design of policy-based composite web services qos monitoring system. *IJCCBS 2,* 1, 79–91.



Amal Alhosban is a Ph.D. candidate in the Department of Computer Science at Wayne State University. She received her Masters degree in Computer Science from Al al-Bayt University, Jordan, and joined there as a faculty member. Since starting her doctoral studies at Wayne State University she has served under various capacities (e.g. Graduate Research / Teaching Assistant, Student Assistant, and Part Time Faculty) in the Department of Computer Science. Her research interests include service computing, reliable distributed systems, semantic Web, data mining and wireless networks. She has published several research papers on these topics, and is an active member of ACM and IEEE.

Khayyam Hashmi is a Ph.D. candidate in the Department of Computer Science at Wayne State University, where he is a member of the Services COmputing REsearch (SCORE) Laboratory. He received his M.S in computer science degree for National University of Computer and Emerging Sciences, Pakistan. His research interests include service computing, distributed systems, semantic web and software process engineering.

Zaki Malik is an Assistant Professor of computer science at Wayne State University, where he heads the Services COmputing REsearch (SCORE) Laboratory. His research interests lie broadly in service computing, reliable distributed systems, Web databases, and semantic integration systems. The focus of the research is on applying service-oriented techniques in these areas to build computer systems that are deployable in practice. He has published several papers in the above areas in top journals and conferences (e.g., VLDBJ, WWWJ, ICSOC, ICWS, etc), and authored a book on Trust Management in Service-oriented Environments. He also serves in various capacities on the committees and boards of database and service-oriented computing conferences and journals. He received the PhD degree in Computer Science from Virginia Tech, and is a member of the IEEE and ACM.

Brahim Medjahed is Associate Professor at the University of Michigan - Dearborn's Department of Computer and Information Science. He received the Ph.D. degree in computer science from Virginia Tech in May 2004. He received the 2004 "Outstanding Graduate Research Award" at Virginia Tech's Department of Computer Science. He also received the Computer Journal Wilkes Award for 2008 (Best paper award). His research interests include service-oriented computing, Web services, data integration, and semantic Web. He authored a book published by Springer on semantic Web service composition. He has published more than 60 papers in international journals and conferences.