SCISPACE
formerly Typeset

Open access • Journal Article • DOI:10.1007/S10845-013-0855-6

QoS ontology for service selection and reuse — Source link

Sophea Chhun, Néjib Moalla, Yacine Ouzrout

Published on: 01 Feb 2016 - Journal of Intelligent Manufacturing (Springer US)

Topics: Differentiated service, Service (business), Service design, Mobile QoS and Service level objective

Related papers:

- Web Services Selection Based on Mixed Context and Quality of Service Ontology
- WS-QoSOnto: A QoS Ontology for Web Services
- Web Services Selection Based on Mixed Context and Quality of
- A Survey and Analysis on Semantics in QoS for Web Services
- Methodologies for selection of Quality Web Services to Develop Efficient Web Service Composition

Share this paper:

View more about this paper here: https://typeset.io/papers/qos-ontology-for-service-selection-and-reuse-1g3oxgrlkg

QoS ontology for service selection and reuse

Sophea Chhun, Néjib Moalla, Yacine Ouzrout

▶ **To cite this version:**

Sophea Chhun, Néjib Moalla, Yacine Ouzrout. QoS ontology for service selection and reuse. Journal of Intelligent Manufacturing, Springer Verlag (Germany), 2016, 27 (1), pp.187-199. 10.1007/s10845-013-0855-6 . hal-01356032

HAL Id: hal-01356032

https://hal.archives-ouvertes.fr/hal-01356032

Submitted on 12 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

QoS Ontology for Service Selection and Reuse

Sophea CHHUN, Néjib MOALLA, Yacine OUZROUT

Université Lumière Lyon2, DISP laboratory, France

{sophea.chhun, nejib.moalla, yacine.ouzrout}@univ-lyon2.fr

Abstract. Web service technologies become popular in software development in all sectors. Even service related standards are defined, they still have limitations to represent the services. Some examples of the limitations are: service registry does not support the QoS properties, web service description language (WSDL) does not allow specifying the QoS properties and there is no common description for defining the web service structure. Our research aims to enhance the service description structure to support the service selection and composition process in order to reduce development costs. The existing resources are analyzed in order to define a Web Service Ontology (WSOnto). Furthermore, a service selection algorithm is proposed for validating the proposed WSOnto. This service selection algorithm considers multiple criteria as inputs: application context, input, output, weight of the QoS performance and service's security properties. Furthermore, a QoS calculation method is proposed to obtain a better QoS values.

Keywords: Business process re-engineering, Ontology, Semantic web service, Service composition, Service discovery, Service Selection, Tracking data, QoS

1 Introduction

The business processes are created from many business tasks joined together by using connectors (parallel, inclusive, exclusive, event based and complex). They are executed on service oriented architecture (SOA). Each business task is performed by a service and a service can be reused by many business applications. Reusing the existing available services reuses development cost. In order to obtain a service for executing a business task, the service selection algorithm is required. The service selection algorithms match between the user's requirements and the services' specifications. A service is specified by its functional and non-functional properties. The functional properties of service describe its provided operations which describe what a service can do. The non-functional properties of service define additional information about the service such as performance, security, and business information [1]. The QoS is a sub part of the non-functional properties of a service. Many services can provide the same functionality. In this case, the QoS is used to select the most suitable service [2]. This proves that using the QoS can improve the re-usability of services. In [1], Khu-

tade and Phalnikar state, "*Neglecting QoS will cause serious problems in software development*".

Web service is described by service description languages such as WSDL (Web Service Description Language) and ontology is used to represent the semantic description of the web services. Different ontology languages are available such as OWL (Ontology Web Language), OWL2 (Ontology Web Language–Second Edition), OWL-S (Ontology Web Language-Semantic) [3] and WSMO (Web Service Modeling Ontology) [4-5]. However, all the current existing standard languages do not support the storing of the non-functional properties of the service and do not define the common structure to represent services. Extend those languages to support non-functional properties are mandatory. Defining a common standard to represent services is also required. For example in [2], the authors extend OWL-S profile ontology to represent QoS vocabulary (business QoS, performance QoS, etc.) and business offers values (offer start time, offer end time, offer type, etc. to support online shopping service system. There is no standard ontology structure to represent services' description and functional and non-functional properties. The existing semantic structures are defined based on specific business application domains such as online shopping, transportation service, online room booking and online restaurant booking. There is no standard, neither for QoS ontology nor on how to calculate the QoS values.

Service registry UDDI (Universal Description Discovery and Integration) or service repository are used to store services [7]. UDDI is not only allowing us to store service information but also provides APIs (Application Programming Interface) to publish and invoke services. However, it does not support semantic matching, it supports only keywords matching. In addition, it does not store Service Non-Functional Properties (SNFP), and SNFPs are important in service selection and composition process. Thus, solutions are required to solve this problem.

Sometime atomic services are unable to do it. In this case, service composition (integration) is required to create a new service by joining many services together. Service selection and composition can be done in static or dynamic way or it is done by syntactic or semantic comparison. Static means that the selection and composition of services are predefined in the programming code; the dynamic solution is done at runtime. Syntactic matching uses keywords matching and does not care about synonym and homonym of the words being matched. However, semantic matching considers the synonyms, homonyms and relationships between words that have been matched. In our research study, we use both dynamic and semantic services matching to solve business process re-engineering problems by reusing existing available services. Reusing existing available services can reduce the development costs (time and money) and provides better control of existing services.

From the existing service selection and composition solutions, based on syntactic or semantic comparison, there are 3 possible ways to express user's requirements: (i) by using keywords (also known as context or goals); (ii) by using only functional properties of services such as input and output; (iii) by using functional and non-functional properties of services. For our research, we consider all the 3 criteria (context, service functional and non functional properties) because it provides better accu-

racy results. However, it also increases the complexity of the solution and execution time.

There are some factors which increase the service selection and composition complexity: (i) large number of available services; (ii) various business application domains; (iii) lack of existing standard for web service description and service registry to express functional and non-functional properties; (iv) many algorithms for service selection and composition.

Actually, our research study aims to support the reuse of existing available services in the re-engineering of business process applications deploying service oriented architecture. A framework is required to create in order achieve our goal. It must provide the capability to perform the automatic service selection and composition in order to obtain the best service that can respond to the requested task. In addition, the framework must be able to generate an executable business process from user's business process specifications by reusing existing services. Furthermore, the services that are used in our model are obtained from the previous deployed business process applications, and they are stored in a common service registry UDDI. The reuse of existing services is proposed to reuse the increasing number of new created services. It also decreases the maintaining problems. Figure 1 introduces the overview of our framework architecture.

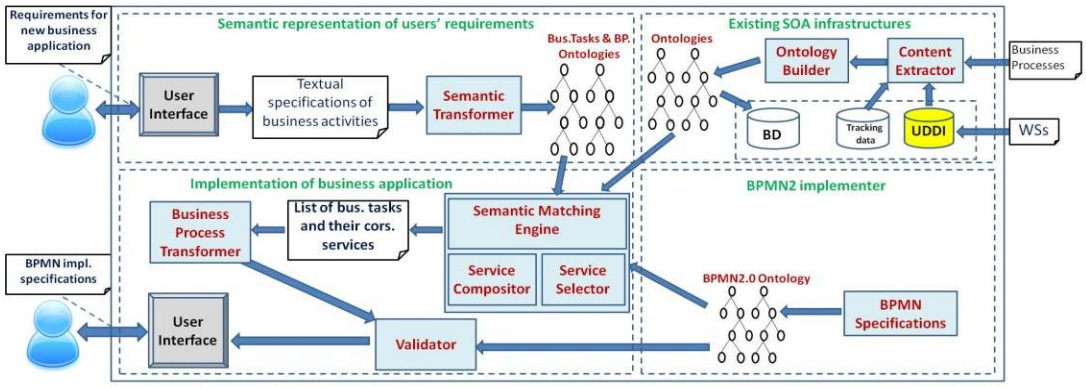

Fig. 1. Framework architecture of the service selection and composition process

Our framework is divided into 4 main modules and each of them is described in detail as follows:

1. **Semantic representation of users' requirements**: from the user's business process specifications, we provide a user interface to define additional information about each business task such as task's description, functional properties (input, output) and non-functional properties (QoS). Then, the module generates different business task ontologies to represent each inputted task of the business process. At the same time, it generates a business process ontology used later for reconstructing the user's inputted business process.

2. **Existing SOA infrastructure**: our model uses the existing services and the services' execution tracking data to build the web service ontology (WSOnto) for representing the available services with their QoS values. Then, it uses the WSOnto to support the service selection algorithm when selecting the best suitable service for executing the user's business task. Furthermore, our model generates another ontology to represent the deployed business processes that can be used in the service composition process. The ontology is selected to represent the published services because UDDI does not allow to store the QoS values and it supports only the keywords matching. The generation of ontologies task to represent the existing resources in our model is responsible by a module called "Existing SOA Infrastructure".

3. **BPMN2 implementer**: BPMN specifications ontology is used to support the reconstructing and the evaluation of the output business processes of our model. It can also detect the contradiction of the generated business processes. In this part, we reuse the existing BPMN 2.0 ontology defined in [8].

4. **Implementation of business application**: this module is responsible for generating the final deployable business processes and it ensures their syntax correctness.

This paper focuses only on the "Existing SOA infrastructure" module. The aim of this module is to define a web service ontology structure to represent services' information by considering their functional and non-functional properties. Another objective of this module is to propose a method to generate an ontology for representing the previous deployed business processes. To define the ontology structure, we analyze the existing services' information resources from UDDI's data, WSDL files and services' execution tracking data from the server. Moreover, we propose a service classification technique and we present a method to calculate the QoS values from services'execution tracking data.

The rest of this paper is organized as follows. Section 2, "Related Works", presents the current existing solutions to our research study. Section 3, "Proposed Solution", introduces our web service ontology structure, the mothod to calculate the QoS performance values, and a service selection algorithm. Then section 4, "Discussion", describes the added value of our solution and the main issues to be tackled. Finally, we finish this paper with a conclusion about our work and introduction to future work.

2 Related Works

In this section, we will first analyze the different algorithms for the Selection and Composition of Web Services (table1). We will also define the QoS attributes and why it is mandatory to use ontologies to facilitate the services selection and composition. From these analyses we will identify the main issues and the needs in terms of semantic selection and composition.

Table 1. Comparison table of matching model

Proposal	Input	QoS attributes	QoS data source	QoS's structure	Algorithm model	Service composition
[2]	OWL-S (functionality, IOPE, QoS, business offers)	8 attributes (quantitative atts.)	OWL-S file (service providers)	ontology	Ontology based (subsumption reasoning + sequencetial)	no
[9]	A vector of QoS values	m QoS attributes (quantitative atts.)	not specified	ontology	Graph (sequential) +Combinatorial	yes
[10]	A vector of QoS values & weight	m QoS attributes (quantitative atts.)	not specified	vector	Graph (sequantial+general flow) + Combinatorial	yes
[12]	Domain information + a vector of QoS	config. 9 dyn + 1 static (cost)	static defined + dynamic monitored (not specify detail how to get value)	class	Combinatorial	no
[19]	weight of QoS attibutes	m attributes (4 attributes in case study)	not specified	ontology	Utility function	no
[17]	Input (data type), output, service category, custom module	no	not specified	ontology	Ontology based (subsumption reasoning)	no
Our	context, input, output, QoS performance's weights, Service's security	6 attributes (quantitive+qualitative)	Services' execution tracking data	ontology	Ontology based (sequential)	no

2.1 Web Services Selection Algorithms

Combinatorial model. In [9], the authors propose 2 algorithms for service selection and composition, combinatorial and graph based algorithm. Combinatorial algorithm adapts Multi-dimension Multi-choice 0-1Knapsack Problem (MMKP) to solve service selection problem. MMKP is defined as following: (i) K groups of elements; (ii) each group has l_i ($1 <= i <= K$) items; (iii) each item is defined by a profit P_{ij} and requires resource $r_{ij} = (r_{ij1}, r_{ij2}, \ldots, r_{ijm})$; (iv) total amount of available knapsack resources $R = (R_1, R_2, \ldots, R_m)$. Knapsack problem aims to select exactly one item from each group by respecting the resource constraint and maximize the total profit. This problem is formulated as:

$$Max \sum_{i=1}^{N} \sum_{j=1}^{l_i} F_{ij} x_{ij}$$

$$Subject\ to\ \sum_{i=1}^{N} \sum_{j=1}^{l_i} q_{ij}^{\propto} x_{ij} \leq Q^{\propto}\ (\propto = 1, \ldots., m)$$

$$\sum_{j=1}^{l_i} x_{ij} = 1$$

$$x_{ij} \in \{0,1\};\ i = 1, \ldots, N;\ \ j = 1, \ldots, l_i \qquad (1)$$

Where $x_{ij} = 1$ if the service is selected from its group and $x_{ij} = 0$ otherwise.

This MMKP problem is mapped to service selection problem as following: (i) K groups are mapped to N service classes; (ii) each item of a K group corresponds to a service in a service class N; (iii) profit of item is identified by utility function; and resource of each item is presented by QoS attributes; (iv) available resources are mapped to user's QoS requirements. Each service class presents a set of services that provides the same functionality to solve a specific task. The same authors [9] divide the QoS attributes into 2 groups, one group requires to maximize their values and the other group requires to minimize them. Actually, this work supposes that from a user's requirement, the system generates many groups of services working together to

answer to the user's requirement. These groups of services are presented by process plans. A process plan is a sequential of services working together to solve a user's requirement. The service utility function that is calculated based on QoS properties and their weight are defined as:

$$F(K) = \sum_{i=1}^{\propto} w_i * \left(\frac{q_{ai}(K)-\mu_{ai}}{\sigma_{ai}}\right) + \sum_{j=1}^{\beta} w_j * \left(1 - \frac{q_{bj}(K)-\mu_{bj}}{\sigma_{bj}}\right) \quad (2)$$

Where α: number of QoS properties that are required to maximize their values; β: number of QoS properties that are required to minimize their values; w: weight of each QoS parameter set by user (0< wi, wj <1); μ and σ are average value and the standard deviation of the QoS attribute for all candidates in the service class; $\sum_{i=1}^{\propto} w_i + \sum_{j=1}^{\beta} w_j = 1$; α+β = m. In short, this algorithm calculates the sum of QoS value of all process plans, then choses the one with maximum value of profit or the utility function value. The algorithm works only for sequential composition. MMKP problem is NP-hard. Yu, et al. [10] introduce an exponential time algorithm called BBLP base on branch and bound method. Branch and bound method is presented in [11]. BBLP generates a search tree in which each node stores a state solution, fixed or free. Fixed means that the service in the service class is selected, and free otherwise. A tree node has 3 state values: (i) fixed utility value Ff produced by fixed classes $S_i \in C, F_f = \sum_{s_i \in C} Fi_{p_i}$; (ii) utility upper bound (Fb): Fb=Ff + FLP. Linear relaxation of sub problems is used to calculate FLP; (iii) branching class (Sg) is the solution of FLP; it is a service class Si with the highest value of xij. BBLP build the search tree by 3 steps. Firstly, for each node find the utility upper bound Fb by linear relaxation of the whole problems. Then, choose a node with the maximum value of Fb and extend the tree through Sg. The selected class is added as new node to the tree's current node. Lastly, repeat the previous steps until all classes are fixed. Each node with the highest value of Ff is the optimal solution. The selected candidate of each class can be found by tracing back to the root of the tree. The same authors in [10] presents heuristic algorithm called WS_HEU which always selecting a feasible solution without thinking the unfeasible one that does not respect to the constraint. This algorithm runs in polynomial function; it iterates in each group to upgrade the current solution. If it cannot fine a better solution, it will use upgrade follow by downgrade method to improve the solution. Feasible solution is a solution that maximizes utility function and does not violent the constraint. From the start, if there is no feasible solution found, algorithm stops. The algorithm is used to select services in one process plan. Therefore, if many process plans matched, it requires applying the algorithm for every plan. Finally, it chooses the one with highest utility value. Yu, et al., [10] propose a solution which support rich composition structures such as parallel, conditional and loops of services. In [12], Cabrera et al. propose the WeSSQoS system which works over multiple web service repositories and QoS resources; it uses QoS values to rank the matched services. This system asks users to provide 2 inputs, domain context (refers to functionality requirement) and a vector of QoS defined by value: a Boolean defines whether QoS attribute is required to be maximized or minimized; another a Boolean defines whether QoS attribute is mandatory or not. The selection algorithm follows 3

steps to choose the most suitable service, normalization, ranking and then priority evaluation. Furthermore, different normalization and ranking algorithms are introduced by the authors.

The combinatorial model is used to rake a set of services that provide the same functionality of service and select the best service based on a utility function calculated from QoS values and their weight. Therefore, before applying this combinatorial model, we must apply another pre-process algorithm first to obtain a set of services that provides the same functionality of the service.

Graph model. Graph algorithm can process with more than one process plans at a time, unlike the combinatorial algorithms. The candidate graph is constructed as follows: (i) each graph node corresponds to a candidate item of the service classes and each node has a profit value (utility function) and a list of QoS (attribute and its value); (ii) if a service in class S_i connects to a service in class S_j, then all services in class S_i are connected to all services in class S_j; (iii) in every links, set the network QoS attributed to it (accumulated QoS values); (iv) add a virtual source node V_s and sink node V_d. V_s is connected to all nodes without incoming links and all nodes that do not have outgoing links are connected to V_d. The QoS attributes of these links are set to 0; (v) calculate the new QoS value of each node by adding the QoS value of its incoming link to its original value. The same accumulation process is applied to the utility function value. An example of service candidate graph is presented in figure 2.

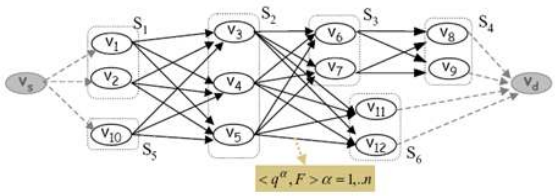

Fig. 2. Service candidate graph (source: [9])

The algorithm passes through the graph in topological short order. Each node stores a number of paths that respect the constraint ($\forall \alpha, q^{\alpha} \leq Q^{\propto}$); each accumulative of QoS value in each node must be less or equal to the value provided by users (one QoS attribute at a time). The algorithm's details are available in [9]; the authors propose MCSP-K by just keeping only K paths in each node.

The Graph solution models the problem as a multi-constraint optimal path (MCOP) problem. It verifies the constraints and selects the maximum utility in each step when passes through each node of the graph from the source node to the sink node. However for combinatorial model, it compares the utility function values of all the process plans to choose the one with maximum value of the utility function and verify the constraint of QoS values.

2.2 QoS Attributes

Many services can provide the same functionality, so QoS could be used to further select the best service among the matched services [13]. Currently, the standard representation of the QoS properties in the service registry is not yet defined. Therefore, defining a structure to represent the published services and their QoS properties is required. The existing research works use ontology to represent services and their QoS properties.

The QoS ontology is usually defined by experts to obtain a comprehensive structure because experts understand the domain application very well. Moreover, QoS ontology is mainly defined for a specific application domain. The QoS properties are categorized into groups according to their characteristic such as performance and security. Khutade and Phalnikar, [1] use ontology to represent QoS properties and categorize them into 3 parts: (i) performance: latency, scalability, throughput, response time, availability, error rate and accessibility; (ii) security: encryption, authentication, accountability, access control, authorization and confidentiality; (iii) Business property: monitoring, reputation, payment method and cost. Chaari, et al. [14] make a case study about transportation service for delivering goods. They divide non-functional properties of the service into QoS properties and context properties. QoS properties are presented into 2 subgroups, execution and security. Context properties are divided into business properties and environment properties. The environment value defines the location of the service provider.

Publishing the services by using the WSDL does not allow specifying the value of QoS. However, the semantic web service description (WSDL-S) allows representing the QoS properties. Another solution is to extend OWL-S profile. For example, D'Mello and V. S. Ananthanarayana, [2] extend OWL-S profile ontology to support the QoS vocabulary that describe the QoS in the level of performance, business and business offers values. The QoS in the level of business offers is detailed by offer start time, offer end time, offer type, etc.

In addition, the values of the QoS can be provided by the service providers or from the services' execution tracking data stored on the server. The service requesters can also provide the feedback on the QoS. The calculation of the QoS values is a difficult task because the QoS values can be quantitative and qualitative. Some of QoS properties are required to be maximized but others are required to be minimized. For example, response time and error rate need to be minimized; but the availability and reputation require to be maximized. The examples of the calculation of the QoS values are presented in [15][16]. Many research works study about defining the QoS properties and categorizing into groups. However, few research works mention about how to obtain and to calculate the QoS values.

In conclusion, using the QoS properties for characterizing the services is required the additional work in storing, publishing and calculating. If we want to consider the QoS properties in the service selection and composition algorithm, we have to consider some issues such as: (i) application domain, is it generic or specific? This gives us a vision on how to define QoS attributes; (ii) how to define a generic and extensible QoS vocabulary? (iii) how to represent QoS properties in service's advertisement,

storing and user's query? (iv) how to obtain QoS values? (v) how to calculate QoS values in service selection and composition processes?

2.3 Ontology Encapsulation for Enhanced Service Specification

Ontology improves the semantic representation of web services and faster the research because of its tree structure. However, the variety of ontology languages and system domains create difficulty in choosing the most suitable ontology language and its data structure. In order to choose an ontology language, the specifications of the ontology languages are required to study such as: data structure, creation purpose, expressiveness, reasoning method, and supported data types.

Ontology defines the hierarchy and the relationship between different concepts. It is used to support service selection algorithm and to provide the degree of subsumption matching [17]. In [18], the author propose the service and domain ontology to represent service's description and programmer's API comments. He uses GATE framework with M-POS and M-DEP to build ontology. At the same time, JAPE rules are used to extract the necessary keywords from service's description and API comments. Tian and Huang [19] build a lightweight domain ontology by using tModel values of the UDDI to hierarchy the ontology. This tModel of UDDI is further sub categorized by categoryBag value. This ontology does not store the equality concepts, but they create a separate synonym table to support it. In [20], the authors propose a service discovery approach base on ontology. They use ontology to semantically represent the user queries, service descriptions and contextual information. The user's query is identified by service type, outputs, inputs and contextual attribute. Their proposed algorithm compares the users' inputs, outputs and service type with the service ontology and data type ontology to identify 3 degrees of matching: precise match, approximate match and mismatch. Then, the algorithm uses the contextual data to rank the result. Another example of service matching based on ontology is defined in [21].

Klein [22] cites many keywords-based and table-based service discovery techniques which are not recommended due to the low quality of the retrieved services.

Therefore, the ontology provides the ability for defining many degrees of matching between 2 concepts and not just only exact matching.

3 Proposed Solution

The existing proposed solutions are mainly considered the QoS values from the service providers. They do not consider the values of QoS obtained from services' execution tracking data on the server. For our proposed solution, the values of QoS performance properties are calculated from the services' execution tracking data on the server. Moreover, a method to obtain better value of the services' execution time is introduced. This method considers the services' execution states to calculate the value of service's execution time.

A web service ontology (WSOnto) is defined to represent the available services, the QoS properties and services' cateogries. The existing research works define the ontology's structure by experts and for a specific domain application. However, our research generates the structure of web service ontology automatically. It uses the tModel values of UDDI to categorize the services into groups.

Then a service selection algorithm that requires multi-criteria as input is introduced to validate the proposed ontology structure. The multi criteria are domain context, service's functional properties, QoS performance's properties and service's security. The service's security properties include the authentication information and the supported encryption methods. For the security reason of the information systems, web services should not be invoked by anyone. It is vital to consider user's access right in the service selection process because it is possible that the result service from the service selection algorithm cannot be invoked. These multi criteria are taken into account to obtain a better accurate and comprehensive results of the service selection algorithm.

This paper focuses on "Existing SOA Infrastructure" module which can be seen in figure 3. This module is responsible for defining a structure to represent the existing services published by many business process applications and to store the service's QoS values. The proposed structure must support with other modules of the global framework presented in the figure 1. Once a service is published, it is stored in the UDDI and then it is passed to the "Content Extractor" component. The "Content Extractor" component extracts useful data, then it forwards those data to the "Ontology builder" component. After the "Ontology builder" component receives those data, it will store them in the ontology. Moreover, the "Content Extractor" component is also used to extract the content the services' execution data (performance value of services) and deployed business processes. The deployed business processes are stored in a new structure because it is later used in the service composition process. Thus, the ontologies are needed. One for storing the services, another one for storing deployed business processes. A scheduler process is created to backup ontologies into a database to prevent any lost. All the data is stored in one database to facilitate the database connections. Ontology is selected to store services' information because of many reasons: (i) the limitations of current UDDI. It does not store QoS values and allow only keyword matching. (2) Ontology is recognized as a semantic representation of services because it defines the relationships between the concepts stored in the ontology. (3) Ontology is a tree structure, so it is supposed to provide better searching time compare to other data structure such as table.

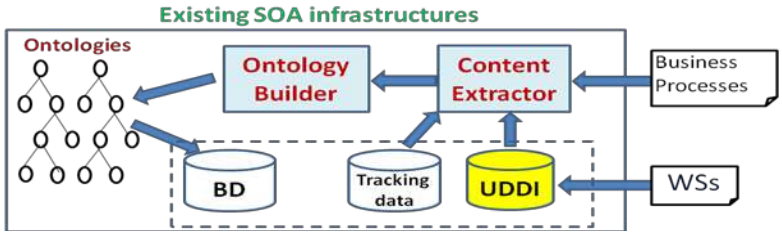

Fig. 3. Existing SOA infrastructures

To achieve our goal, we divide the work into 2 steps. First, define an ontology's structure to represent services' information. Then define a service selection algorithm to validate the proposed structured of ontology.

3.1 Web Service Ontology Building

It is vital to study about existing resources and to define what we need to store in ontology before defining its structure. We have UDDI's data, WSDL files, tracking data. In UDDI, some information can be found such as service's owner (called business entity), service's information and service's accessing constraints. These data can be retrieved by UDDI's APIs. Figure 4 presents service related information extracted from UDDI. Not all the information is shown in the figure 4 in order to make it visible. The other additional information can be found such as bindingTemplate identified by access point (usetype and text description), tModel and categoryBag. CategoryBag is referenced by one or many Keyreferenced and each of them is defined by tmodelKey, name and value. The functional (input, output, pre-condition and effect).

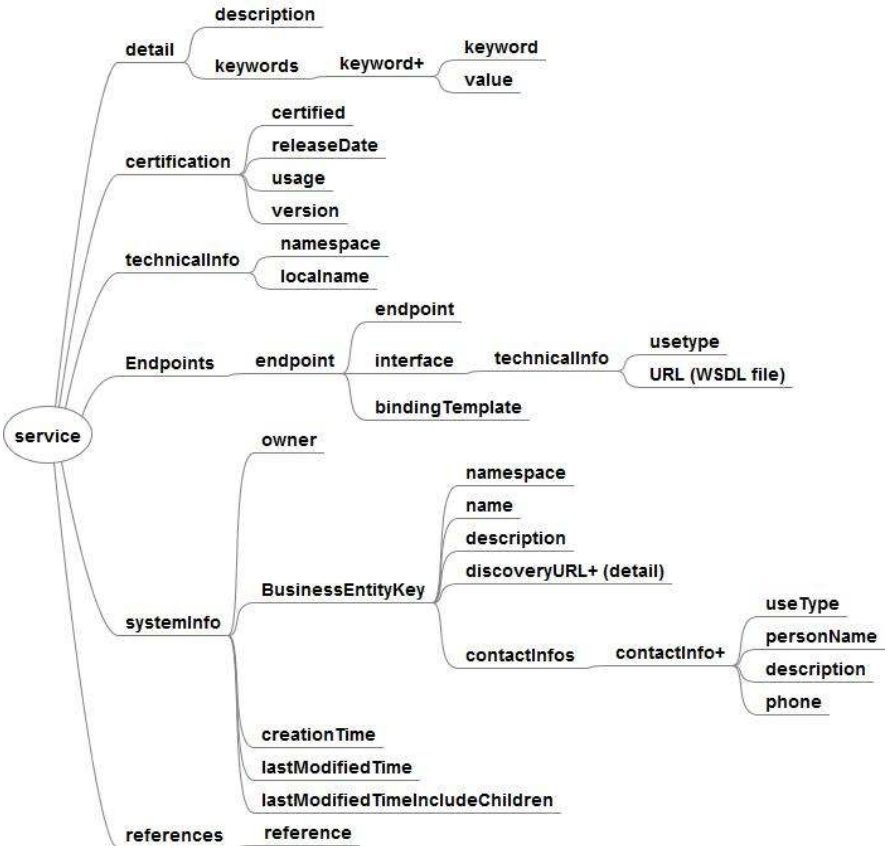

Fig. 4. Service hierarchy tree

The tracking data can be obtained from the server at the level of services (component) or service's operations. Table 2 presents an example of tracking data from the oracle SOA suite at the level of service's operations. This table shows the tracking data of different instances of services and services' operations. It provides the information about composite's name (composite_name), service's name (component_name), service's operation (operation_name), execution start time (begin_time), execution end time (end_time), execution duration (duration_in_second) and execution state (state). The execution state identifies whether a service is successfully executed or not. Moreover, some addition information can be extracted from the table such as the total number of calls and number of failures of the services. Then, the availability value can be calculated from these two values.

Table 2. Tracking data

COMPOSITE_NAME	COMPONENT_NAME	OPERATION_NAME	BEGIN_TIME	END_TIME	DURATION_IN_SECOND	STATE
AssignMappingProject	BPELProcess1	Assign	22-FEB-13 07.05.26.457000000 PM	22-FEB-13 07.05.26.482000000 PM	0.025	5
AssignMappingProject	BPELProcess1	Assign	22-FEB-13 07.05.35.837000000 PM	22-FEB-13 07.05.35.861000000 PM	0.024	5
AssignMappingProject	BPELProcess1	Assign	22-FEB-13 07.05.28.902000000 PM	22-FEB-13 07.05.28.918000000 PM	0.016	5
AssignMappingProject	BPELProcess1	Assign	22-FEB-13 07.05.09.844000000 PM	22-FEB-13 07.05.09.860000000 PM	0.016	5
bpel-101-HelloWorld	HelloWorldProcess	sendMessage	22-FEB-13 06.57.05.898000000 PM	22-FEB-13 06.57.05.909000000 PM	0.011	5
bpel-101-HelloWorld	HelloWorldProcess	sendMessage	22-FEB-13 06.57.10.734000000 PM	22-FEB-13 06.57.10.742000000 PM	0.008	5
bpel-101-HelloWorld	HelloWorldProcess	sendMessage	22-FEB-13 06.55.51.882000000 PM	22-FEB-13 06.55.51.889000000 PM	0.007	5
bpel-106-CallingWebServices	BPELCurrencyProcess	process	22-FEB-13 07.10.47.889000000 PM	22-FEB-13 07.10.48.519000000 PM	0.63	10
bpel-106-CallingWebServices	BPELCurrencyProcess	process	22-FEB-13 07.10.41.350000000 PM	22-FEB-13 07.10.41.956000000 PM	0.606	10

Figure 5 presents our proposed web service ontology's structure. This web service ontology (WSOnto) is composed of 2 parts. The first part presents services' categories and the second part presents services' information. Service's category refers to the value of tModel in the UDDI. Every time when the service providers publish their services, they need to specify the service's category value. The WSOnto is described in detail as follows:

- The first level (depth=1) of WSOnto represents services' categories. Each service category contains a set of services and a list of keywords. These keywords can be extracted from the UDDI.
- A service is identified by its functional and non-functional properties (QoS), and some other properties such as businesskey, servicekey, url (of WSDL) and businessname (service provider).

- The functional properties of service are defined by input and output. The input and output are specified by data-type, name and value.
- QoS is divided into 2 subgroups, performance and security. The service's performance defines how well a service was executed. The service's security defines the supported security mechanisms by a service and users' authentication information and users' access right. Service's performance is specified by availability, executionTime and totalcalled of service. Service's security is identified by encryption (supported encryption method), authentication (users' authentication information), and authorization (users' access right information). The availability, executionTime and totalcalled are defined by 2 properties, value and maximize. Maximize property defines whether the attribute requires to be maximized or minimized. It has Boolean value of 0 and 1.
- A service contains one or many operations and each operation is identified by its functional properties and QoS performance.

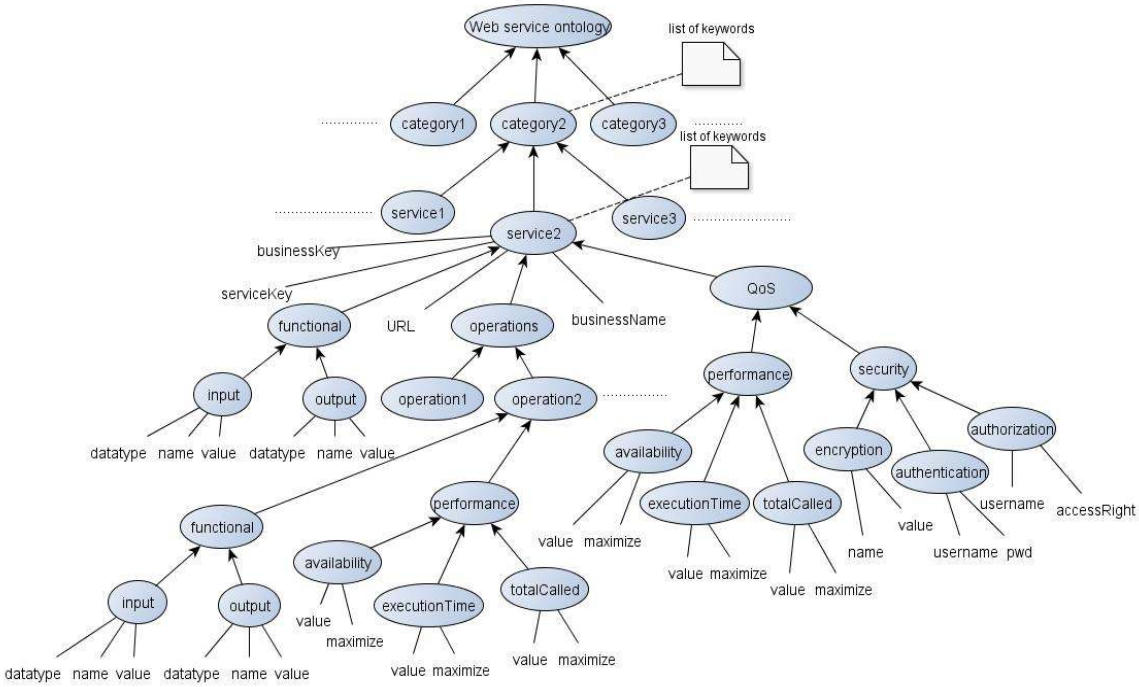

Fig. 5. Web service ontology

3.2 QoS value

We are not only focusing on building the ontology structure to represent web services and defining a service selection algorithm to validate the structure, but we propose

also a solution to improve the calculation of QoS values. This yields to a better accurate result of the service selection algorithm. Usually, the duration of execution time of the services is calculated as the average value of the execution time of all service's instances. This way does not really show the reality value of execution time of service because; the execution time of the failure service's instance should not be taken into account.

Table 3. Execution time of a service's instances

Service	ExecutionTime	……	Status
instance 1	t1		1
instance 2	t2		0
…..	…..		1
…..	…..		1
instance n	tn		0

Table 3 presents an example of the execution time of a service's instances with the status equal to fail (status=0) or success (status=1) when the service's instances were called. The final value of execution time of a service, S_{ex}, is defined in equation (3).

$$S_{ex} = \frac{\sum_{i=1}^{n} t_i St_i}{n} \quad (3)$$

Where n is the total number of instances of a service; S_{ex} is the duration of the execution time of a service; St defines the status of the service's instance. The calculation function of the service's execution time and Table 3 provide an example of execution time attribute case, but it can be applied for other attributes if needed. For the execution time of the service's operations can be calculated by using equation (3) as well.

A script is scheduled to run in order to update the QoS values of services and their operations that are stored in the WSOnto.

Our proposed solution considers only the QoS values from the server and does not consider the QoS values from the service providers. It is also a solution to consider the QoS values from the server and the service providers. However, if we want to consider the QoS values from the service providers, OWL-S or WSDL-S must be used to describe the web services. Table 3 provides an example of the calculation of the service's execution time and service's availability value from the server and the service provider. The calculation is described as follows: (i) if the value of the availability and execution time of service are provided by the server and service provider, then the final value is the average value of the values obtained from the server and the service provider; (ii) if one of the server or service provider does not specify the QoS values, then the final QoS values are taken from the one that has value; (iii) otherwise, they are set to zero.

Table 4. Calculation the QoS values

	Availability				ExecutionTime			
Server	Vsa	Vsa	0	0	Vse	Vse	0	0
Service Provider	Vpa	0	Vpa	0	Vpe	0	Vpe	0
	Avg(Vsa+Vpa)	**Vsa**	**Vpa**	**0**	**Avg(Vse+Vpe)**	**Vse**	**Vpe**	**0**

3.3 Service Selection Algorithm

A service selection algorithm is proposed to validate the structure of WSOnto. This algorithm considers only the selection of atomic service to perform the users' requirements. The user's requirements are defined by: (i) context that is defined by a set of keywords. (ii) Service's functional properties (an input and an output); (iii) service's security value (username and password); (iv) weight of each attribute of the QoS performance values. The algorithm works in 5 steps sequentially as shown in figure 6.

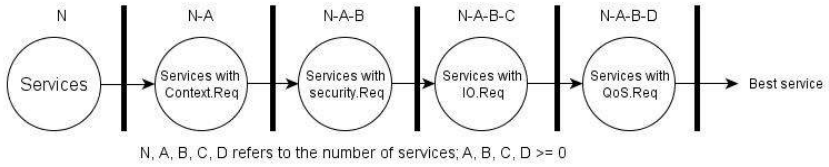

Fig. 6. Service selection process

The pseudo code of the service selection algorithm is presented as follows:

```
Service Selection (Req, WSOnto) {

  /* create a tree contains services that are verified
with user's context specification */

  OntoTree = filterContext(context, WSOnto);

/* remove all nodes that are not verified with the user's
security constraints */

  OntoTree = filterSecurity (security, OntoTree);

/* remove all nodes that are not verified to the IOs con-
straints*/

  OntoTree = filterFunctionality (IOs, OntoTree);
```

```
  /* rank and select the most suitable service base on
the QoS Performance values*/

  Result = selectBestService (performance, OntoTree);

}
```

The algorithm can be described in detail as follows:

(a) filterContext: this method builds a new tree that stores all services verified with user's context (keywords) from web service ontology (WSOnto). It passes through the web service ontology to compare the user's context with service category's keywords, and then compares the user's context with the service's keyword to select the services that verify user's context constraint. All the validated services are copied to OntoTree.

(b) filterSecurity: this method removes all service nodes that are not verified with QoS security constraints (username and password). When absent the inputted values from the user, filterSecurity still check the service's authentication value. If the service's authentication is empty, the service node is kept. Otherwise, it is removed. The service's authentication is empty means the service can be invoked by anymore.

(c) filterFuntionality: this mothod has OntoTree, inputs and outputs as its input. It passes through all the services stored in OntoTree one by one and compares service functional properties with the inputs and outputs provided by users. This test supposes that the input and output are defined by data type and name. filterFunctionality compares the service's outputs with user's outputs and service's inputs with user's inputs. All the services that are not verified with the service's functional property constraints are removed from the OntoTree.

(d) selectBestService: this method calculates the utility values of all services in OntoTree, then it ranks the result by the utility values descendant. Then it outputs a service with the maximum value of utility. The calculation of utility value is followed equation (2).

4 Discussion

The designed web service ontology (WSOnto) represents the existing available services, QoS values, and the services' operations. WSOnto is used to eliminate the limitations of UDDI and WSDL. The limitations of UDDI concern the semantic matching and it cannot store the non-functional properties of services. For WSDL, it does not allow introducing the QoS values. Moreover, OWL is used to build the WSOnto because it is a W3C recommendation, expressiveness, support reasoning, and follows object oriented programming. In WSOnto, each service and service's category link to a list of keywords. These keywords are used to solve homonym problem. The WSOn-

to stores also some important service's attributes allow accessing other additional information of services in the UDDI.

After defining the WSOnto, the equation (3) is defined to calculate the service's execution time by taking into account the value of service's execution state. The execution time is a Boolean value of 0 and 1. The final value of service's execution time is calculated by the sum of all instances of service's execution time multiply with the value of the service's execution state, then divide with the number of service's instances. This calculation produces the average value of service's execution time for only successfully executed service's instances. This calculated method produces a better result than just taking the average value of the execution time of all service's instances.

Then a service selection algorithm is proposed to validate proposed WSOnto's structure. The service selection algorithm uses multi criteria to select the best suitable service to perform a user's requirement. One important criteria of the service selection algorithm are the service's security properties. The service's security properties are defined by the encryption methods, user's authentication and user's authorization information. Fewer researchers consider the service's security properties as a selection criterion in their service selection algorithms. If the service's security properties are not taken into account in the service selection algorithm, the algorithm can output the results that cannot be invoked by the service requesters. In addition, the QoS performance's weight is also one of the multi criteria of our proposed service selection algorithm. The QoS performance's weight is used to easily allow users specifying their preference of QoS performance's values. The QoS performance's weight allows to obtain better accurate result by avoiding the big gap between different values of the QoS performance properties. For example the total number of service's calls, a service is called just for a few times while the other one was called hundreds of times. Thus, without considering the weight can cause a big effect on the value of the utility function.

Taking into account multi criteria in the service selection algorithm improves the output results, but it also increases the complexity of the algorithm.

5 Conclusion and Future Work

In this paper, we define a web service ontology (WSOnto) to represent existing available services, services' categories, and the QoS values. WSOnto stores also the services' operations with their functional properties and QoS values. The tModel of UDDI values is used to automatically categorize the services into groups. Moreover, each service and service's category are linked with a list of keywords that can be used in the service selection algorithm. We proposed a method to better calculate the value of services' execution time by considering the failure or the success state of services' execution. The services' execution tracking data on the server is used to calculate the QoS performance values. An example table of the tracking data from the oracle SOA server is introduced. Then a service selection algorithm is proposed to test the pro-

posed web service ontology. It takes multi-criteria as input. The services' security is also an input criteria of our proposed service selection algorithm.

We want to make use of the full advantage of ontology, therefore, we aim to represent the users' requirements in ontology structure and perform the matching between users' requirements ontologies and the WSOnto to obtain the most suitable service to perform user's requirements. Therefore, our future work is to define the ontologies to represent users' requirements and propose service composition algorithm. Then complete the other parts described in our global framework.

Acknowledgements: This project has been funded with support from the European Commission (EMA2-2010- 2359 Project). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. A. Pradnya Khutade and B. Rashmi Phalnikar, "QOS BASED WEB SERVICE DISCOVERY USING OO CONCEPTS," *International Journal of Advanced Technology & Engineering Research (IJATER)*, vol. 2, no. 6, 2012.
2. D. A. D'Mello and V. S. Ananthanarayana, "Semantic Web Service Selection Based on Service Provider's Business Offerings," *IJSSST*, vol. Vol. 10, no. No. 2, Mar. 2009.
3. D. Martin, M. Burstein, and et al., "OWL-S: Semantic Markup for Web Services," vol. 22, 2004.
4. H. H. Wang, N. Gibbins, T. R. Payne, and D. Redavid, "A formal model of the Semantic Web Service Ontology (WSMO)," *Information Systems*, vol. 37, no. 1, pp. 33–60, 2012.
5. J. de Bruijn, C. Bussler, J. Domingue, and et al., *Web Service Modeling Ontology (WSMO)*. 2005.
6. C. Von Riegen, "UDDI version 2.03 data structure reference," 2002.
7. T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, and J. Munter, "UDDI Version 3.0," *Published specification, Oasis*, vol. 5, pp. 16–18, 2002.
8. C. Natschläger, "Towards a BPMN 2.0 Ontology," presented at the Business Process Model and Notation, 2011, pp. 1–15.
9. T. Yu and K. J. Lin, "Service selection algorithms for composing complex services with multiple QoS constraints," *Service-Oriented Computing-ICSOC 2005*, pp. 130–143, 2005.
10. T. Yu, Y. Zhang, and et al., "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, p. 6, 2007.
11. S. KHAN, "Quality adaptation in a multisession multimedia system: Model, algorithms and architecture," Department of ECE, University of Victoria, 1998.
12. O. Cabrera, M. Oriol, X. Franch, L. López, J. Marco, O. Fragoso, and R. Santaolaya, "WeSSQoS: A Configurable SOA System for Quality-aware Web Service Selection," *arXiv preprint arXiv:1110.5574*, 2011.
13. S. Neelavathi and K. Vivekanandan, "An Innovative Quality of Service (QOS) based Service Selection for Service Orchrestration in SOA," *International Journal of Scientific and Engineering Research, 2 (4)*, 2011.
14. S. Chaari, Y. Badr, F. Biennier, C. BenAmar, and J. Favrel, "Framework for web service selection based on non-functional properties," *International Journal of Web Services Practices*, vol. 3, no. 2, pp. 94–109, 2008.

15. M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, “A scalable approach for qos-based web service selection,” in *Service-Oriented Computing–ICSOC 2008 Workshops*, 2009, pp. 190–199.
16. D. Schuller, U. Lampe, J. Eckert, R. Steinmetz, and S. Schulte, “Cost-driven Optimization of Complex Service-based Workflows for Stochastic QoS Parameters,” in *Web Services (ICWS), 2012 IEEE 19th International Conference on*, 2012, pp. 66–73.
17. M. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Mühl, and K. Geihs, “Ranked matching for service descriptions using owl-s,” in *Kommunikation in Verteilten Systemen (KiVS)*, 2005, pp. 91–102.
18. M. Sabou, “Building Web Service Ontologies,” 2006.
19. Y. Tian and M. Huang, “Enhance discovery and retrieval of geospatial data using SOA and Semantic Web technologies,” presented at the Expert Systems with Applications, 2012.
20. T. Broens, S. Pokraev, M. Van Sinderen, J. Koolwaaij, and P. Dockhorn Costa, “Context-aware, ontology-based service discovery,” *Ambient Intelligence*, pp. 72–83, 2004.
21. M. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Mühl, and K. Geihs, “Ranked matching for service descriptions using owl-s,” in *Kommunikation in Verteilten Systemen (KiVS)*, 2005, pp. 91–102.
22. A. Bernstein and M. Klein, “Towards high-precision service retrieval,” *The Semantic Web—ISWC 2002*, pp. 84–101, 2002.