An Extension of OWL-S with Quality Standards

Stéphane Jean, Francisca Losavio, Alfredo Matteo, Nicole Lévy

An Extension of OWL-S with Quality Standards

Stéphane Jean*, Francisca Losavio†, Alfredo Matteo† and Nicole Levy‡
*Laboratory of Applied Computer Science
LISI/ENSMA and University of Poitiers
BP 40109, 86961 Futuroscope Cedex, France
jean@ensma.fr
†Laboratorio MoST, Centro ISYS, Escuela de Computacion, Facultad de Ciencias
Universidad Central de Venezuela, Venezuela
francislosavio@gmail.com, almatteo@cantv.net
‡University of Versailles
St-Quentin en Yvelines, France
nlevy@prism.uvsq.fr

***Abstract*—With the increasing amount of Web Services available on the Web, Web Services discovery issues are becoming increasingly important. Since current Web Services standard technologies (e.g. UDDI) only provide syntactic descriptions of Web Services (mainly their signatures), semantic discovery approaches based on ontologies (e.g. OWL-S) have been developed. These approaches lead to more precise descriptions of Web Services functionalities, but they provide few mechanisms to capture non functional aspects of Web Services collectively referred as Quality of Services (QoS). To fill this gap, some works have proposed Web Services discovery approaches based on QoS ontologies. However these approaches do not take into account existing standards about software quality and the relationships that can be established between them. Yet, these standards could be used as a shared understanding between services providers and customers and thus, they would ease the Web Services discovery process. In this article we first propose an extension of OWL-S to describe QoS according to one or many quality standards. Then, we develop an approach based on this extension of OWL-S to improve the Web Services discovery process. This approach is based on an extension of SPARQL that simplifies expression of Web Services discovery queries. Relationships between standards are used to return Web Services even if they are described with quality properties defined in an other standard that the one used to express queries. Finally, non functional requirements can be expressed as user preferences. Thus, they can be used to rank Web Services fulfilling functional requirements during the Web Service discovery process.**

I. Introduction

As Web Services (WS) technologies gain popularity, the design of an increasing number of software applications is based on existing WS. However, selecting the convenient WS for a given application is not an easy task. Indeed this selection process requires to find those WS that provide the exact functionalities required by the designed application (*functional requirements*). Moreover, these functionalities must be accomplished with a certain degree of quality (*non functional requirements*). In this paper, we only consider non functional requirements related to quality. In the domain of services, the term *Quality of Service* (QoS) is used, i.e. the set of quality properties that a WS must fulfill.

The WS selection process may be accurate only if services are described very precisely. However, current techniques based on SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description Discovery and Integration) only rely on a syntactical description of WS interfaces. Recently, semantics approaches based on ontologies have been developed. In particular, the OWL-S ontology [1] has been defined to describe semantically WS. Nevertheless, if OWL-S is a rich model for describing WS functionalities, it offers few QoS concepts to express their qualities.

Different standards have been defined by institutions and organizations such as ISO/IEC [2], [3] or the W3C [4], to describe precisely quality properties characterizing software products. The wide usage of these standards in the domain of WS should help WS consumers and providers to communicate with a common language on WS quality and thus, to find an agreement. However, even if different approaches have been developed on QoS to improve WS discovery, few works have taken into account the existing standards on software products quality. As a consequence, evaluating quality of a WS described according to one of these approaches is difficult. Moreover, even if quality standards were used, the availability of several standards may lead to difficulty for evaluating a WS if it is described with another standard that the ones knows. Thus it is necessary to represent and exploit relationships that can be established among the quality properties definitions and metrics defined in the various standards.

In this paper, we propose an extension of OWL-S for QoS-based WS description using existing software product quality standards. Our approach is based on the ontology we have proposed in [5]. This ontology can be used to represent quality properties defined for a given domain (for example, the WS domain), as they have been defined by different standards. Moreover, relationships that can be established among these properties (for example, the properties that are equivalent in two standards), can also be specified. We first show how OWL-S can be extended with this ontology. Then we study WS discovery capabilities offered by this extension. In order to facilitate the WS discovery process, we propose (1) an

extension of the SPARQL query language [6] that simplifies the expression of a WS discovery query, (2) a set of rules to exploit relationships that can be established among quality properties, eventually defined by different standards and (3) an approach to express non functional requirements as user preferences and thus, to order services satisfying functional requirements according to quality preferences.

The rest of this paper is organized as follows. In the next section, we review related QoS-based approaches on semantic WS description and discovery. In section III the ontology on which our approach is based is presented, and in section IV we show how it can be used to extend OWL-S. Section V details the interest of this extension for WS discovery. Section VI presents an implementation we have made for validating our approach; advantages and limitations of our proposition are also discussed. Finally, we conclude in section VII and describe perspectives opened by this work.

II. Related Work

Numerous ontology-based approaches have been proposed to deal with QoS, such as [7], [8], [9], [10]. In a recent state-of-the-art [11], Tran.et al. show that each existing approaches focus on particular aspects of the QoS representation and management but does not provide a complete solution with all desirable features. This section describes related work on QoS ontologies which are mainly based on OWL-S such as the proposed approach in this paper. For each one of them, the QoS aspects on which the described ontology focuses on, is pointed out.

OWL-Q [7] is an extension of OWL-S that provides complex and extensible mechanisms to associate metrics to quality properties. Indeed, it includes concepts to represent simple and complex (derived from others) metrics, and even to add new metrics. Moreover, complex algorithms are provided to compare and exploit these metrics in queries.

QOS-MO [8] defines concepts to represent QoS of WS described in OWL-S. QOS-MO focuses on the description of interactions between WS consumers and providers. Indeed, this ontology defines the concept of `QoSContract` relating `QoSOffered` to `QoSRequired` in order to represent an agreement on quality of WS between WS consumer and provider.

onQoS [9] associates QoS description to OWL-S WS profiles. This ontology proposes a powerful data type system to define quality properties values. These values can then be evaluated using metrics, scales or mathematical formula.

DAML-QoS [10] proposes an extension of DAML-S (antecedent of OWL-S) for QoS. Using this ontology, one can define constraints that can be associated to quality properties. For example, the concept of `QoSPrecondition` can be used to define conditions which must be fulfilled by WS consumers to obtain the QoS specified by providers.

We note that none of the previous approaches deal with the representation and exploitation of software products quality standards and relationships that can be established between them. Indeed, even if these ontologies could be used to represent quality properties defined by a given standard, origin of these properties could not be preserved. For example, these approaches can not represent the quality model (structured set of characteristics that describes the quality of a software product [2]) of the standard in which a given quality property has been defined. Yet, this information could be useful to understand the described quality of a WS. Moreover, these approaches define few relationships between quality properties. Equivalence and subsumption relationships between classes are available in OWL but they do not apply on instances and thus on quality properties. These relationships are needed to establish a mapping between quality properties defined in the various standards considered. This mapping could be useful to solve problems related to the diversity of existing standards on software products quality.

Furthermore, considering the WS discovery process, these approaches have mainly developed algorithms exploiting data types and metrics associated to quality properties values. Expression of a WS discovery request using a query language is barely considered. Moreover, non functional requirements are used in these approaches to filter WS. Yet, non functional requirements are often used by users to rank WS fulfilling functional requirements. All these observations have driven us towards the development of a QoS approach centered on quality standards.

III. Ontology for Software Quality Standards

The quality-driven WS discovery approach proposed in this paper is based on the *Ontology for Software Quality Standards (OSQS)* described in [5]. In this section, we present its main components.

A. Software Products Quality Standards Considered

Defining precisely a quality property such as `Security` is very difficult. Nevertheless, clarifying the meaning of a quality property is necessary to help WS providers and customers finding an agreement. Several software products quality standards have been defined to solve this problem. They provide precise definitions of many quality properties with associated metrics when possible. The OSQS ontology presented below allows integrates the following standards.

Web Services Architecture (WSA) [4].

The WSA reference architecture has been defined by the Web Services Architecture Working Group to guarantee the interoperability of WS-based applications. The specification of this architecture defines a set of quality properties or goals to evaluate the compliance of a WS to this architecture. This specification defines seven quality goals: `Interoperability`, `Reliability`, `WWW Integration`, `Security`, `Scalability and Extensibility`, and `Team Goals`. For example the `Security` goal of a WS has two requirements: (1) `Protection from threats` of a WS across distributed domains and platforms, and (2) `Privacy protection` (enable privacy policy on WS) for the consumer of a

WS. Protection from threats is further refined into sub-goals such as Threat of accessibility attacks, Authentication of the parties, Authentication of authorship of data and Authorization. Note that metrics are not provided by this standard.

The ISO/IEC 9126-1 standard [2]

The ISO/IEC 9126-1 standard defines quality properties for software products. These quality properties are organized in a hierarchy, where the higher abstraction level is constituted by six main characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. These characteristics are refined until measurable characteristics, called attributes, are obtained. For example, Functionality implies existence of a set of functions that satisfy particular needs. It is refined into sub-characteristics such as Suitability, Accuracy, Interoperability and Security. These sub-characteristics can be further refined according to the specific application domain. The ISO 9126-1 hierarchy of quality characteristics, sub-characteristics, attributes and metrics constitutes the quality model of an application in a given domain. Contrary to WSA, this standard is not specific to the WS domain: ISO 9126-1 is a framework that must be adapted to the WS domain eliminating the non applicable characteristics or adding new sub-characteristics if necessary.

The ISO/IEC 13236 standard [3]

The ISO/IEC 13236 standard concerns quality of services. In this context, the term *service* has a very broad sense; for example it includes, but is not limited to, the provisioning of interactions, the processing and information repository functions by entities, objects, applications, processes, communication services, etc. The ISO 13236 standard defines a terminology and several concepts on QoS to provide a common language for WS providers and customers. Moreover, it introduces a set of characteristics, mechanisms and metrics for describing, specifying and managing QoS requirements. More precisely, it proposes categories of high level quality characteristics. A characteristic represents some aspect of the QoS of an application, service or resource that can be identified and quantified. These characteristics are mainly time-related, capacity-related, reliability-related, safety-related or security-related. An example of characteristic is Security. Security can be refined into Access control (protection against unauthorized access to a resource) and Data protection (protection against unauthorized access to data); the metrics for both properties are described as a value derived from the application of an access control or a data integrity policy, respectively. Actually, the probability of failure of the policy (real value) and also the presence or not of the policy (boolean value) can be considered acceptable metrics for these attributes.

The diversity of software quality standards leads to terminology variability. To solve this problem, we have proposed in [5] an ontology to represent these various standards with their relationships. This ontology, on which our approach is based, is described in the next section.

B. Description of the OSQS Ontology

The main components of the OSQS ontology are shown in Figure 1.

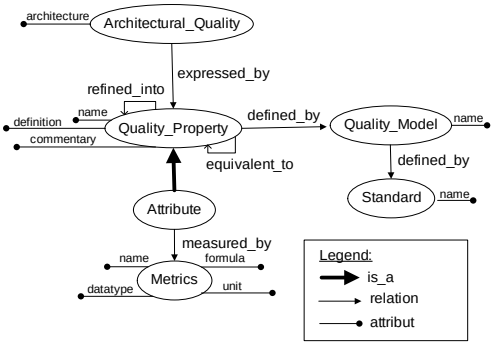

Fig. 1. Extract of the OSQS ontology

- **Architectural_Quality**: represents the quality of an architectural style for a family of applications. This quality is expressed by a set of quality properties that must be fulfilled. SOA is an example of architectural style fulfilling the mandatory quality properties of interoperability, security, availability and maintainability.
- **Quality_Property**: represents a quality characteristic which can be refined into one or many other quality characteristics, which can or cannot be measured. Security is an example of a quality property. According to WSA, it is refined into Protection from threats and Privacy policy; Protection from threats is further refined into the measurable quality attribute Authorization. The relation equivalent_to may be used to establish correspondences between quality properties defined by different standards. For example, according to the ISO 13236 and WSA standards, ISO 13236 Access_control is equivalent to WSA Authorization, because an access control implies an authorization service and vice-versa. To simplify Figure 1, we have not shown all relationships that can be established between quality properties. These relationships will be detailed in section V-C.
- **Attribute**: represents a quality property that can be measured. For example, the measure of Access_control defined in ISO 13236 is the value obtained from the application of an established access control policy. Notice that properties defined in a particular standard without metric can be characterized by the metric of an equivalent property. For example, even if the WSA standard does not provide any metric, the WSA Authorization property can be associated to the boolean metric (shown in Figure 2) because it is equivalent to the Access_control of ISO 13236. Notice also that some metrics are implicit.

For example, `Privacy_policy` is associated to a boolean metric because a service may or may not have a privacy policy.

- **Metrics**: represents a metric expressed by a value of a given data type. This value can be characterized by a unit and can be computed by a formula. For example, a measure for `Data_protection` - refinement of the ISO 13236 `Security` characteristic - can be the probability of failure of the protection. This metric has real values (between 0 and 1); it has no unit of measure, nor formula (thus, its values are defined by users). Due to space limitations, we do not detail representation of units of measure. Metrics definition depends on the implementation model chosen. For example, the PLIB ontology model [12] natively supports metrics, whilst OWL [13] does not provide built-in unit constructors. In the second case an ontology such as the Measurement Units Ontology[1] may be used as an extension.
- **Quality_Model**: represents a set of quality properties specifying the quality of a software product.
- **Standard**: represents an agreement on a certain view of software quality. ISO 13236 and WSA are examples of quality standards in the WS domain.

Figure 2 shows an instantiation of OSQS using previous examples. In this example, the SOA architecture is characterized by quality properties from the standards ISO 13236 and WSA. For readability, we have not represented the complete quality models of these standards; we have just shown the higher level properties and pre-fixed them with the corresponding standard. These quality properties can be refined into attributes. For example, `Security` of ISO 13236 is refined into attributes `Data_protection`, measured by a real value and `Access_control`, measured by a boolean value. `WSA_security` can be refined into `Privacy_policy` and `Protection_from_threats`; the `Privacy_policy` and `Authorization` are attributes measured by a boolean value.

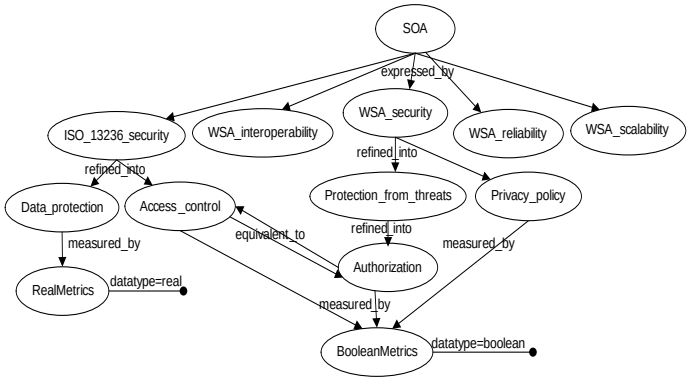

Fig. 2. Instantiation example

[1]http://idi.fundacionctic.org/muo/muo-vocab.html

We have used the OSQS ontology presented in this section to extend OWL-S.

IV. Proposed Extension of OWL-S

OWL-S [1] is an ontology defined by researchers from different organizations that aims to declare and describe WS. The goal of this ontology is to facilitate the dynamic discovery and invocation of WS and also their composition. If this ontology includes many concepts to describe precisely functional characteristics of a service, we show in what follows that it provides few QoS concepts.

A. Limitations of OWL-S for QoS

In OWL-S, a service is described according to the following three dimensions.

- **ServiceProfile**: describes the functionalities offered by a service in terms of inputs, outputs, preconditions, parameters and results.
- **ServiceModel**: describes how to use a service, detailing its parameters and indicating the step by step process leading to the results from these parameters.
- **ServiceGrounding**: describes how to access a service. In particular, it specifies a communication protocol and message formats.

The OWL-S specification states that the `ServiceProfile` can be used to specify quality of WS. Indeed, the `ServiceProfile` is associated to a set of `ServiceParameter` to characterize a service by pairs (criterion, value). Nevertheless, this very simple model has some limitations. First, the description of quality properties (expressing the goals of non functional requirements) can not be represented with this simple model. Thus, the definition given by the standard defining a quality property can not be specified. Yet, this definition may be very useful for users interactions. Secondly, this simple model can not be used to represent relationships between qualities properties. In particular, it is not possible to refine or establish correspondences between them. However, these relationships can be useful to exploit the specified quality of services. For example, the work presented in [14] shows that these relationships are useful to identify the relevant quality criteria for a software product audit. In section V-C we show that they can also be useful for WS discovery. Finally, the model proposed by OWL-S to represent QoS does not include concepts to associate metrics to quality properties, especially complex metrics based on mathematical formula or units of measure. However, most of quality properties need this modeling feature.

In order to overcome these limitations, we propose to extend OWL-S with the OSQS ontology introduced in section III-B.

B. Proposed Extension

Our proposed extension of OWL-S is presented in Figure 3. In OWL-S, the `ServiceProfile` characterizes the service functionality, detailing elements such as input

(ServiceInput) and output (ServiceOutput) of a service. As it has been already pointed out, ServiceProfile is also described by a set of ServiceParameter composed of a name (serviceParameterName) and a value (sParameter). Our OWL-S extension is based on the specialization of ServiceParameter by the Quality_Property class of the OSQS ontology. The remaining elements of the OSQS ontology are kept unchanged, except for the property name associated to Quality_Property which disappear, since it is now inherited from the ServiceParameter class, and for Architecture_Quality which is suppressed since OWL-S is only concerned by a specific style of architecture: the SOA architecture.

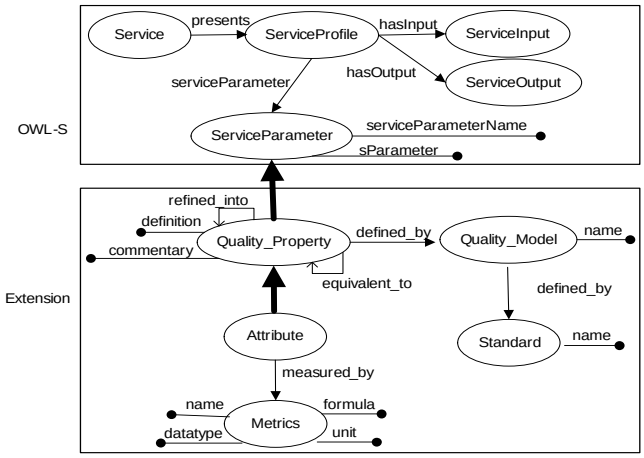

Fig. 3. Proposed extension of OWL-S with OSQS

In this section we have first identified several limitations of OWL-S for describing non functional aspects of WS and then, shown how this ontology can be extended with the OSQS ontology considering the software products quality standards. In the next section we present the interest of this extension for the WS discovery process.

V. WS Discovery According to Quality Criteria

The OWL-S extension proposed in this paper can be useful for the different actors involved in the diffusion and utilization of WS. For example, a WS registry administrator can use it to select one or several quality standards that must be used to describe the non functional aspects of the services declared in its registry. Then, service providers will have to register their services specifying values of quality properties defined in these standards. This process can be assisted using our extension of OWL-S by displaying definitions and comments of the quality properties used.

In this paper we focus on the interest of the proposed extension of OWL-S for the customer searching for a WS. We begin by describing the query capabilities offered by this extension and by pointing out their limitations.

A. Query Capabilities Offered by our OWL-S Extension

Since OWL-S is an ontology, an ontology query language, such as SPARQL [6] or OntoQL [15], can be used to search for a WS.

Example. The following SPARQL query[2] search for the services having as input a credit card that use an access_control as it is defined in the ISO 13236 standard.

```
SELECT ?service
 WHERE {
  ?service rdf:type Service
  ?service presents ?profile
  ?profile hasInput <http://cordio.bs/CreditCard>
  ?profile serviceParameter ?parameter
  ?parameter defined_by ?qualityModel .
  ?qualityModel defined_by <http://iso.org/ISO13236>
  ?parameter serviceParameterName "access_control"
  ?parameter sParameter ?value
  FILTER (?value = true) }
```

Explanation. The first triple of the WHERE clause introduces the ?service variable iterating through the whole set of services. This variable is used in the SELECT clause to return all the services satisfying other triples. Since the properties needed to express the request are defined by the service profile, the second triple uses the presents property to introduce it through the ?profile variable. Next, the third triple checks that the service has a credit card as input. The fourth triple introduces the ?parameter variable in order to iterate over the quality properties describing the service. The next two triples restrict the quality properties considered to those defined in the ISO 13236 standard quality model. The last three triples check that the service is described by the quality property named (property serviceParameterName) access_control and that it satisfies this property (value true).

This example shows that our extension of OWL-S allows users to search for a service on the basis of quality properties defined by a particular standard. More complex queries could be written to retrieve values of all quality properties describing the service with their definitions provided by the corresponding standards. Moreover metrics could be used in queries. For example, it could be useful to retrieve units of measure of quality property values.

However, we identify that the capabilities offered by our proposed extension of OWL-S present the following three shortcomings.

- Queries are written encoding the proposed extension of OWL-S. Moreover, quality properties are not used in queries as other properties. Indeed, before testing values of quality properties, it is first necessary to retrieve them through the sParameter property.
- Results of queries are produced without considering relationships that can be established between quality properties. For example, the previous query will not

[2]For conciseness we omit namespaces in queries.

return services that fulfill the Authorization quality property of WSA, even if this quality property is equivalent to the ISO 13236 access_control.

- All conditions expressed in queries on quality properties are strict constraints; i.e. a service will only be returned if it satisfies them. Yet, a user will in general look in a first place for services accomplishing the required functionalities (strict constraints on the functional requirements), and will secondly look for those satisfying the required quality properties (soft constraints on non functional requirements).

We detail in next sub-sections our proposed solutions to overcome these drawbacks.

B. SPARQL Extension for Quality-based WS Discovery

As we have pointed out in the previous section, the SPARQL syntax is not adapted for searching for a WS according to quality criteria. More precisely, we identify the following three problems.

- The proposed OWL-S extension is encoded in queries. For example, the query of the previous example is written indicating that quality properties are related to a service profile through the serviceParameter property. As a consequence, writing such a query requires a deep knowledge of our OWL-S extension. Moreover, this query is tight to this extension and thus to its possible evolution.
- Quality properties are used differently from other ontology properties. Indeed, in order to test a value of a quality property, it is necessary to retrieve the value using the sParameter property before testing it. For homogeneity and simplicity sake, it would be better to use them as other properties.
- Functional and non functional requirements are mixed in queries. Indeed, the WHERE clause of our example contains both triples for functional requirements (the one that uses the hasInput property) and triples for non functional requirements (the subsequent triples). For readability and modularity of the query, these two types of triples must be distinguished.

In order to overcome these problems, we propose an extension of the SPARQL language.

1) Syntax of the Proposed SPARQL Extension: Our aim is to extend the SPARQL language to facilitate the search for WS according to quality criteria. Since this requirement is very specific, the proposed extension should be modular and, for sake of homogeneity, it should keep a syntax similar to the one of SPARQL. In order to fulfill these requirements, we propose to add a new clause, named QUALITY, to the SPARQL language. This clause has a syntax[3] similar to the WHERE clause:

```
QualityClause  ::= QUALITY '{' TriplesQuality+ '}'
TriplesQuality ::= VarOrTerm PropQualityOrProp Object
```

[3]This syntax is simplified and is based on notations used at the URL http://www.w3.org/TR/rdf-sparql-query/ to define the SPARQL grammar. In particular, the complete syntax makes it possible to use the FILTER and OPTIONAL operators in the QUALITY clause.

Explanation. The QUALITY clause is composed of a set of triples (subject, predicate, object). The subject can be a variable or a term (VarOrTerm) as defined in the SPARQL syntax. Thus, a quality property can be subject of a triple which could be useful to retrieve its description. The predicate can be a property of an ontology or a quality property (instance of the Quality_Property class, or of its subclass Attribute). The object is similar to the definition given in the SPARQL grammar.

This SPARQL extension can be used to rewrite the previous query example as follows:

```
SELECT ?service
 WHERE {
  ?service rdf:type Service
  ?service presents ?profile
  ?profile hasInput <http://cordio.bs/CreditCard> }
 QUALITY {
  ?profile access_control "true"
  access_control defined_by ?qualityModel .
  ?qualityModel defined_by <http://iso.org/ISO13236>
 }
```

Explanation. As this example shows, our SPARQL extension distinguishes clearly functional (in the WHERE clause) and non functional (in the QUALITY clause) requirements. The first triple of the QUALITY clause shows that a quality property can be used as a usual property. The next two triples remain similar to the initial query.

2) Semantics of the Proposed SPARQL Extension: If the syntax of the WHERE and QUALITY clauses is similar, their interpretation is quite different. The following rule shows the interpretation of a triple (s, p, o), where p is a quality property:

$$(\texttt{s p o}) \wedge (\texttt{p type Quality\_Property}) \Rightarrow (\texttt{s serviceParameter ?param}) . (\texttt{?param serviceParameterName p}) . (\texttt{?param sParameter o})$$

Explanation. If p is a quality property, the triple (s,p,o) must be interpreted as a set of triples. The first two triples search for p among the quality properties using its name (value of the property serviceParameterName). The third triple indicates that the object of the triple (o) corresponds to the value of the sParameter property. Note that a similar interpretation is made if a quality property is used as a subject of a triple.

The following example illustrates the previous rule:

Example. When the quality property access_control is used as a usual property, the following interpretation is made:

$$(\texttt{?profile access\_control "true"}) \Rightarrow (\texttt{?profile serviceParameter ?param}) . (\texttt{?param serviceParameterName "access\_control"}) . (\texttt{?param sParameter "true"})$$

The other rules we have defined concern metrics that can be associated to quality attributes. The following rule shows the interpretation of a triple (s, p, o) where o is a value characterized by a data type (notation: ^^datatype). We suppose in this rule that p is a quality property (more precisely an attribute) interpreted according to the previous rule.

(s p o^^type)
⇒ (s p o) . (p measured_by ?m) . (?m datatype type)

Explanation. If the value o is characterized by the data type type, the metrics associated to p is used to check that this value is really defined according to this data type. This interpretation is done by expanding the initial triple into three triples. The first triple checks that the literal value (without data type) is associated to the subject s by the property p. The next two triples check that the quality property p is measured by a metric (retrieved by the measured_by property) characterized by (datatype property) the data type type.

Example. The following example shows the interpretation of the access_control attribute characterized by the xsd:boolean data type.

(?profile Access_control "true"^^xsd : boolean)
⇒ (?profile access_control "true") .
(Access_control measured_by ?m) .
(?m datatype "xsd : boolean")

In our approach, we have chosen to manage units of measure following a commonly used approach[4]. This approach consists in using units of measure as data types (with the same notation ^^unit). Thus, a rule similar to the one presented above describes a triple (s, p, o), where o is characterized by a unit of measure.

(s p o^^unitOfM)
⇒ (s p o) . (p measured_by ?m) . (?m unit unitOfM)

Explanation. The rule is similar to the previous one but it is now based on the unit of measure given by the unit property. The query engine uses namespaces (xsd for data types, mymw for units) to distinguish a unit of measure from a data type.

Example. The following example shows the interpretation of the maxTimeByTransaction attribute characterized by the second unit of measure.

(?profile maxTimeByTransaction "1"^^mymw : second)
⇒ (?profile maxTimeByTransaction "1") .
(maxTimeByTransaction measured_by ?m) .
(?m unit "mymw : second")

We have presented our SPARQL extension to improve expression of queries searching for WS according to functional and non functional requirements. In the next section we propose an approach to improve results of these queries by exploiting relationships that can be established between quality properties.

C. Exploitation of Relationships between Quality Properties

Many types of relationships can be established between quality properties. We consider the types of relationships established by Kruchten [16] and, more precisely, their interpretations given in [14]. Nevertheless, in opposition to these works that use them mostly for software products audits, we exploit them to improve the search for WS.

1) Relationships between Quality Properties: In section III-B we have introduced the equivalent_to relationship, indicating that two quality properties are equivalent. This relationship is particularly useful to establish correspondences between quality standards. In addition, we consider four additional relationships: constrains, subsumes, forbids and conflicts. In opposition to the equivalent_to relationship, these relations are only applied to quality properties evaluated with a boolean value. These relationships are defined as follows:

- constrains. If X constrains Y, the quality property Y may be satisfied only if X is also satisfied. For example, authentification[5] constrains authorization, if we consider that if an application does not require an authentication of users, it cannot perform an access control.
- subsumes. If X subsumes Y and the quality property X is satisfied then Y is also satisfied. For example, dataEncryption subsumes digitalSignature, if we consider that when an application performs data encryption, it necessarily uses numerical signatures.
- forbids. If X forbids Y and the quality property X is satisfied, then Y cannot be equally satisfied. For example, portability forbids codedInC, if we consider that a portable application cannot be coded in the C programming language.
- conflicts. If X conflicts Y then X forbids Y and Y forbids X. For example, portability conflicts codedInC, if we also consider that when an application is coded in C, it cannot be portable.

As we show in next section, these relationships can be used to facilitate the WS discovery process.

2) Interpretation of Relationships between Quality Properties: The different types of relationships we have presented above, can be used to perform some reasoning, i.e. deduce new information from existing information. The deduced information can be exploited during query processing to return relevant results that could not be returned without the reasoning.

In this section we present reasoning capabilities enabled by each type of relationships. These reasoning capabilities are

[4]see https://forge.morfeo-project.org/wiki_en/index.php/How_to_use_MUO for a detailed explanation.

[5]defined also in the WSA standard as a refinement of Protection_from_threats

defined by deductive rules; an example of application is given for each rule.

equivalent_to :

$$(\texttt{prop1 value v}) \wedge (\texttt{prop1 equivalent\_to prop2}) \Rightarrow (\texttt{prop2 value v})$$

Explanation. This deductive rule states that if a quality property prop1 has a value v and is equivalent to a quality property prop2, then it can be deduced that prop2 also has the value v.

Example. If access_control and authorization are equivalent, the following rule can be written.

$$(\texttt{access\_control value true}) \wedge (\texttt{access\_control equivalent\_to authorization}) \Rightarrow (\texttt{authorization value true})$$

constrains :

$$(\texttt{prop1 value false}) \wedge (\texttt{prop1 constrains prop2}) \Rightarrow (\texttt{prop2 value false})$$

Explanation. This deductive rule states that if a quality property prop1 is not satisfied and constrains a quality property prop2, then it can be deduced that prop2 is not satisfied either.

Example. If authentication constrains authorization, the following rule can be written.

$$(\texttt{authentification value false}) \wedge (\texttt{authentification constrains authorization}) \Rightarrow (\texttt{authorization value false})$$

subsumes :

$$(\texttt{prop1 value true}) \wedge (\texttt{prop1 subsumes prop2}) \Rightarrow (\texttt{prop2 value true})$$

Explanation. This deductive rule states that if a quality property prop1 is satisfied and subsumes a quality property prop2, then it can be deduced that prop2 is also satisfied.

Example. If dataEncryption subsumes digitalSignature, the following rule can be written.

$$(\texttt{dataEncryption value true}) \wedge (\texttt{dataEncryption subsumes digitalSignature}) \Rightarrow (\texttt{digitalSignature value true})$$

forbids :

$$(\texttt{prop1 value true}) \wedge (\texttt{prop1 forbids prop2}) \Rightarrow (\texttt{prop2 value false})$$

Explanation. This deductive rule states that if a quality property prop1 is satisfied and forbids a quality property prop2, then it can be deduced that prop2 is not satisfied.

Example. If portability forbids codedInC, the following rule can be written.

$$(\texttt{portability value true}) \wedge (\texttt{portability forbids codedInC}) \Rightarrow (\texttt{codedInC value false})$$

conflicts :

$$(\texttt{prop1 conflicts prop2}) \Rightarrow (\texttt{prop1 forbids prop2}) \wedge (\texttt{prop2 forbids prop1})$$

Explanation. This deductive rule states that if a quality property prop1 is in conflict with a quality property prop2, then prop1 forbids prop2 (previous rule) and vice-versa.

Example. If portability and codedInC are in conflict, the following rule can be written.

$$(\texttt{portability conflicts codedInC}) \Rightarrow (\texttt{portability forbids codedInC}) \wedge (\texttt{codedInC forbids portability})$$

In this section we have shown that quality properties relationships can be used to make deductions and thus to improve query results. These relationships can be applied between quality properties belonging to different standards. Thus, a service may be returned even if it is described with another standard that the one used to express the query. In the next section, we show that the WS discovery process could also be improved by expressing quality criteria as user preferences.

D. Expression of Non Functional Requirements as User Preferences

In general, functional requirements are considered as the most important criteria when searching for a WS. They must be satisfied. In contrast, non functional requirements are often soft constraints. They are used to rank WS fulfilling functional requirements. Nevertheless, most of existing approaches do not distinguish functional requirements from non functional requirements. As a consequence, queries based on these approaches do not return WS fulfilling functional requirements that satisfy only partially non functional requirements. Thus, if users express their non functional requirements, it often results in queries that do not return any answer.

In the last years, several work have been conducted on the notion of user preferences in the context of ontologies [17], [18], [19], [20]. In these approaches user preferences are defined according to ontologies. These preferences may then be used in queries as soft constraints. Thus, these approaches are well suited for WS discovery according to non functional requirements. However, they must be extended so that non functional requirements could be expressed as user preferences. We have chosen to extend the work of Tapucu et al. [20] because it proposes a rich and extensible model to define user preferences and an extension of SPARQL and OntoQL to exploit them. In the next section we briefly present this approach.

1) The Used Preference Model: The main elements of the preference model used in our approach is shown in Figure 4. In this model, each preference is identified by an URI (`Preference_URI`). The different types of preferences that we use to express non functional requirements are the following.

- `Numeric_Pref`: corresponds to preferences expressed by a numeric value (`numberValue`). For example, a preference for 4 stars hotels can be expressed with a `Numeric_Pref`.
- `Fuzzy_Pref`: corresponds to preferences expressed by a probability value. For example, `Fuzzy_Pref` can be used to express that probabilities of preferring 2, 3, and 4 stars hotels are respectively 0.1, 0.2, and 0.7.
- `Boolean_Pref`: corresponds to preferences expressed by a list of boolean properties whose values are preferred to be true. For example, a preference on hotel rooms having TV, air conditioner and Wifi can expressed using `Boolean_Pref`.
- `Interval_Pref`: corresponds to preferences expressed by a minimum and a maximum value. For example, `cheap` and `expensive` can be preferences expressed on the cost of an hotel room. Using `Interval_Pref`, `cheap` can be characterized by the interval of prices [45..60] and `expensive` by [90..100].
- `Enumerated_Pref`: corresponds to preferences expressed by an enumeration of preferred instances of an ontology. For example, a preference for `cheap` hotels can be expressed by an `Enumerated_Pref` on instances `{Hotel(Formule1), Hotel(Premiere)}`.

The left part of Figure 4 shows the link between preferences and ontologies. This relationship is established by the `Pref_Link` entity. `Pref_Link` associates a preference to a class or a property of an ontology. Classes and properties are represented by the `Property_Or_Class` entity.

Preferences defined using the model presented above may be used in queries. Indeed, the approach of Tapucu et al. [20] proposes an extension of the SPARQL and OntoQL query languages with a new clause named `PREFERRING` to take into account preferences. For example, if a customer has a preference for `cheap` hotels, the following SPARQL query can be used to help him find its preferred hotels:

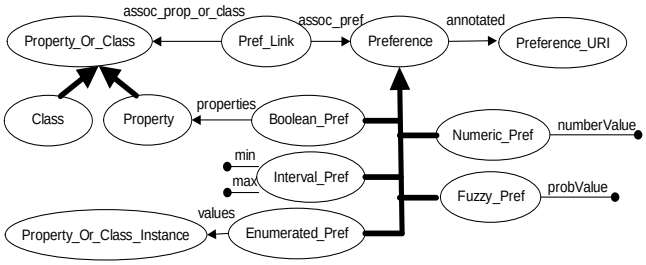

Fig. 4. Extract of the used preference model

```
SELECT ?name ?price
WHERE { ?h type Hotel
        ?h name ?name
        ?h price ?price }
PREFERRING { cheap }
```

Explanation. The `PREFERRING` clause specifies preferences used in the query. These preferences are identified using labels. They are interpreted according to the type of preference they belong to. In this example, `cheap` is an `Interval_Pref` preference linked to the `price` property and characterized by the [45..60] interval. Thus, hotels having rooms whose price belongs to this interval will be returned as first results of this query. Next results will be those hotels having rooms whose price are close to this interval.

As we show in the next section, we have adapted the preference model presented in this section to our OWL-S extension.

2) Preferences on Quality Properties: The preference model presented in the previous section can be used to express preferences on classes and properties. Thus, it can be used to associate preferences to WS (instances of the `Service` class). However, it can not be used to associate preferences to quality properties as it may be done with ontology properties. Indeed, quality properties are instances of `Quality_Property` and thus they are not considered as usual properties. As a consequence, we propose to extend this model to express non functional requirements as user preferences. Figure 5 presents in italic elements that we have added to this model and their links to elements of the initial model. The three main modifications of the initial model are the following.

- In order to associate a preference to a quality property, `Pref_Link` is linked to `Quality_Property` (part of our OWL-S extension) through the `assoc_qualityProp` property.
- In order to define a boolean preference on quality properties, `Boolean_Pref` is linked to `Quality_Property` through the `qualProperties` property.
- In order to define an enumerated preference on quality properties values, `Enumerated_Pref` is linked to `Quality_Property_Value` (set of all quality properties values) through the `qualProp_values` property.

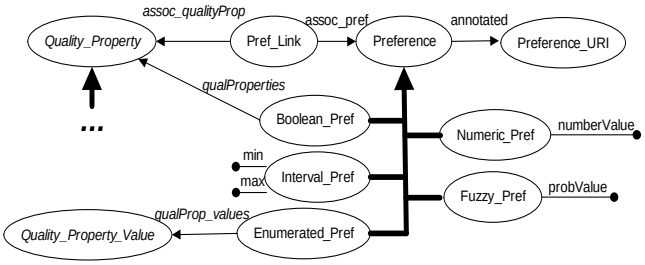

Fig. 5. Extension of the preference model

Figure 6 illustrates the proposed extension of the preference model. Two examples of preferences associated to quality properties are represented as instances of this extension. The first preference is named `Fast`. This preference is an instance of `Interval_Pref` (preference defined by an interval) linked by the `L1 Pref_Link` to the `avgTimeByTransaction` quality property. It defines that services considered as `Fast` are those services having an average time by transaction between 10 and 20 milliseconds. The second preference is a boolean preference named `Secured`. This preference is associated to all services (`Service`) through the `L2` link. It defines that `Secured` services are those services satisfying the `authorization` and `dataEncryption` quality properties.

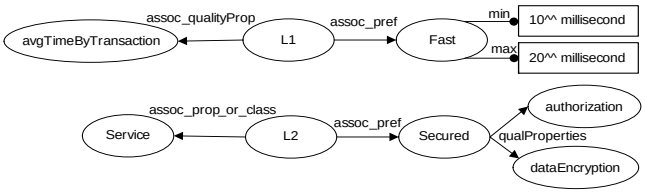

Fig. 6. Example of preferences on quality properties

Preferences associated to quality properties could be specified in queries using the `PREFERRING` clause as it is defined by Tapucu et al. [20]. However, our aim is to propose a modular approach for the management of non functional requirements. Thus, usual preferences must be clearly distinguished from preferences on quality properties. As a consequence, we have modified the syntax of the `PREFERRING` clause in the following way.

```
preferring-clause ::= PREFERRING '{' boolean-expr '}'
boolean-expr ::=
                boolean-term
              | boolean-expr OR boolean-term
boolean-term ::= boolean-factor
               | boolean-term AND boolean-factor
boolean-factor ::= [ NOT ] preference-identifier
     | [ NOT ] QUALITY preference-quality-identifier
```

Explanation. The syntax of the `PREFERRING` clause defines boolean expression (with `AND` and `OR` operators) on preferences identifiers (`preference-identifier`). We have added (part in italics) the possibility of using preferences identifiers on quality properties (`preference-quality-identifier`). These preferences on quality properties are distinguished from preferences on usual property by using the `QUALITY` keyword. Moreover, this keyword simplifies the processing of queries.

The following query illustrates this syntax. It searches for services having as input a credit card, knowing that `Secured` and `Fast` services (represented in Figure 6) are preferred.

```
SELECT ?service
 WHERE {
  ?service rdf:type Service
  ?service presents ?profile
  ?profile hasInput <http://cordio.bs/CreditCard>
 PREFERRING {
   QUALITY Secured AND QUALITY Fast
 }
```

Explanation. Non functional requirements can be expressed as user preferences using our extension of the `PREFERRING` clause. Since the `Secured` and `Fast` preferences are associated to quality properties, they are preceded by the `QUALITY` keyword.

The interpretation of preferences on quality properties is similar to the one defined for preferences on classic properties (detailed in [20]). For example the preference `Fast` is interpreted by the predicate `avgTimeByTransaction >= 10 ^^ millisecond AND avgTimeByTransaction <= 20^^millisecond`. Moreover, we have detailed in section V-B how a quality property could be used as a classic property. Thus, we have defined all necessary elements to interpret preferences defined on quality properties.

VI. IMPLEMENTATION AND EVALUATION

In previous sections we have presented our proposed extension of OWL-S and developed an approach that uses this extension to facilitate WS discovery. In this section we detail the implementation of this approach we have done to validate it and discuss advantages and limitations of our proposition.

A. Implementation of our Approach on the OntoDB Database.

The approach presented in this paper could be implemented on any ontology repository equipped with a SPARQL implementation. Ontology-Based Databases (OBDB) [21] are examples of such systems, which relay on database technology, and thus benefit from their advantages (e.g. scalability or concurrence management). As a consequence, in order to validate our approach, we have chosen to implement it on the OntoDB OBDB [22], which also supports an implementation of the preference model previously discussed. In this section, the different steps followed to achieve this implementation are described.

1) Import of the Extended OWL-S Ontology in OntoDB: OntoDB has been initially designed to support PLIB ontologies. However, this OBDB is also equipped with an OWL ontology import module. As a consequence, we have first extended the OWL-S ontology, as described in section IV, using

the well known editor of OWL ontologies: Protégé[6]. Then, we have used the import module of OntoDB to upload it into this OBDB. More precisely, the OWL-S ontology is decomposed into four OWL files[7]: `Service.owl`, `Profile.owl`, `Process.owl` and `Grounding.owl`. Since our extension is related with the OWL-S profile, we have used the `Profile.owl` file.

2) Extension of SPARQL with the QUALITY Clause: OntoQL is the query language associated to OntoDB. This language has been implemented on OntoDB by translating it into SQL. OntoDB also has a SPARQL implementation that consists in translating a SPARQL query into an OntoQL query. We have chosen to interpret the `QUALITY` clause as specified in section V-B, while translating it into OntoQL. This approach is illustrated in Figure 7.

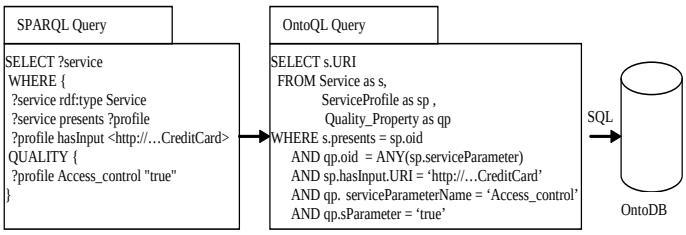

Fig. 7. Interpretation of the SPARQL `QUALITY` clause

Explanation. The initial SPARQL query is presented on the left side of Figure 7 and its translation in OntoQL on the right side. The `WHERE` clause of the SPARQL query is translated into OntoQL by a join between the `Service` and `ServiceProfile` classes and by a predicate in the `WHERE` clause involving the `hasInput` property. The `QUALITY` clause is interpreted by an other join with the `Quality_Property` class and new predicates in the `WHERE` clause to retrieve the quality property named `Access_control` that has the value `true`. This interpretation encodes the rewriting rules expressed in section V-B.

3) Implementation of Rules Exploiting Quality Properties Relationships: OntoDB is not equipped with an inference engine (non deductive database). Thus, the rules specified in section V-C to interpret quality properties relationships can not be implemented directly on OntoDB. As a consequence, we have used the trigger mechanism to implement them. For example, the rule related to the `equivalent_to` relationship is implemented by a trigger on the table storing values of quality properties. This trigger has the following pseudo-code:

```
When a value v of a quality property p1
                is inserted or updated
If p1 is equivalent to a property p2 then
      Update the value of p2 set it to v
```

The other rules have been implemented in the same way.

[6]http://protege.stanford.edu/

[7]available at: http://www.ai.sri.com/daml/services/owl-s/1.2/

4) Extension of the Preference Model: The preference model presented in section V-D1 has been implemented on OntoDB with the OntoQL language. More precisely, an OntoQL script creates all the elements of this model into OntoDB. We have modified this script to take into account the extension defined in section V-D2. For example, the following OntoQL statement modifies the boolean preferences `Boolean_Pref` so that they can be applied to quality properties:

```
ALTER ENTITY #Boolean_Pref
  ADD qualProperties REF(Quality_Property) ARRAY
```

For queries, the SPARQL query engine of OntoDB has been modified to interpret preferences expressed on quality properties. This interpretation being similar to the one done for classic properties, we have mainly reused existing code.

B. Advantages and Limitations of our Approach

Compared to other proposed QoS ontologies (detailed in section II), the originality of our OWL-S extension is to explicitly represent quality standards and the relationships that can be defined among them. In order to exploit this feature, we propose an extension of SPARQL to facilitate WS discovery. This SPARQL extension clearly separates the expression of functional requirements and non functional requirements. Moreover, it takes into account relationships that can be established between quality properties using a set of deductive rules. Notice that the establishment of equivalences between quality properties requires a great amount of expertise on quality standards; the OSQS ontology extending OWL-S contributes to organize this expertise as part of the domain knowledge. Finally, if necessary, the proposed SPARQL extension can be used to express non functional requirements as soft constraints (user preferences) in queries. Thus, users can use functional requirement to search for all services fulfilling their needs and then rank them using non functional requirements.

Since our OWL-S extension focuses on quality standards, on some other aspects, this extension is less expressive than other QoS ontologies. For example, OWL-Q [7] proposes a richer system of metrics and units of measurement than the one of our extension. Indeed, this system includes relationships between metrics, conversions among units, definition of new units, etc. Similarly, WSMO-QoS [23] do not assume that the value of a quality property is fixed all the time. Indeed, some quality properties require to update their values periodically and some applications may need to record the successive values of a given quality property. As a consequence, the valid period of a quality property value can be specified in WSMO-QoS. This aspect that has an impact on WS discovery is not natively supported by our approach. As future work, we plan to integrate specific features of other QoS ontologies into our proposed OWL-S extension.

VII. Conclusion

In this paper we have proposed an extension of OWL-S to facilitate WS discovery. This extension consists in a

set a concepts allowing users to represent quality properties describing a WS as they are defined by the main standards related to software product quality. Using this extension, users may not only record the standard in which a quality property has been defined but also represent relationships that can be established (for example, equivalence or subsumption) between quality properties. Compared to other proposed QoS ontologies, taking into account quality standards is the main specific feature of our approach.

To go a step further, we have studied the interest of the proposed OWL-S extension for WS discovery. Based on this extension, we have shown that queries can be written to retrieve WS according to quality criteria expressed using a given quality standard. However, we have highlighted that the SPARQL syntax is not adapted to express these queries. As a consequence, we have proposed an extension of this query language to clearly distinguish expression of functional and non functional requirements in queries. Moreover, we have noticed that relationships between quality properties were not used to improve queries results and thus, we have defined a set of deductive rules to exploit them. These rules allow a user to retrieve a service even if it is described with other quality properties than the ones used to express the query. Finally, we have considered the fact that non functional requirements are often not crucial for users but used to prioritize services responding to particular functional requirements. As a consequence, we have developed an approach for expressing non functional requirements as user preferences. This approach is based on a rich preference model defining different types of preferences. The defined preferences on WS qualities can be used in queries and are clearly distinguished from other usual preferences.

This work leaves open several perspectives. In this paper, we have not considered composition of services. Indeed, only single WS can be returned as the result of a query. However, a composition of WS may fulfill users functional and non functional requirements. Thus, it would be interesting to study how our approach can be adapted and extended to composition of WS. We also plan, as we have already pointed out, to extend our OWL-S extension with specific features of other QoS ontologies. Finally, it would be necessary to develop more usable tools, allowing the different actors that use WS to benefit from our approach.

Acknowledgment. This work is supported by the Postgrado en Ciencias de la Computación de la Facultad de Ciencias and the Consejo de Desarrollo Científico y Humanístico (CDCH) of the Universidad Central de Venezuela, the CDCH ADIRE Project No. PG-03-7310-2008/1, the UCV-Proyecto nro. 2009000624 of Fonacit, and finally the O2M Project, LISV, France.

REFERENCES

[1] D. Martin, M. Burstein, E. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services*, World Wide Web Consortium, 2004, http://www.w3.org/Submission/OWL-S/.

[2] ISO/IEC9126-1, "Quality characteristics and guidelines for their use (1, 2)," International Organisation for Standardisation / International Electrotechnical Commission, Tech. Rep., 2001.

[3] ISO/IEC13236, "Quality of service: Framework," International Organisation for Standardisation / International Electrotechnical Commission, Tech. Rep., 1999.

[4] D. Austin, A. Barbir, C. Ferris, and S. Garg, "Web Services Architecture Requirements," *W3C Working Group Note 11 February 2004*, 2004, http://www.w3.org/TR/wsa-reqs/.

[5] F. Losavio, A. Matteo, and N. Levy, "Web Services Domain Knowledge with an Ontology on Software Quality Standards," in *Proceedings of the Third International Conferences on Internet Technologies and Applications (ITA 2009)*, 2009, pp. 74–85.

[6] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," *W3C Recommendation 15 January 2008*, 2008, http://www.w3.org/TR/rdf-sparql-query/.

[7] K. Kritikos and D. Plexousakis, "OWL-Q for Semantic QoS-based Web Service Description and Discovery," in *Proceedings of the 1st Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR 2007)*, 2007.

[8] G. F. Tondello and F. Siqueira, "The QoS-MO ontology for semantic QoS modeling," in *Proceedings of the 2008 ACM symposium on Applied computing (SAC 2008)*. ACM, 2008, pp. 2336–2340.

[9] E. Giallonardo and E. Zimeo, "More Semantics in QoS Matching," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007)*, 2007, pp. 163–171.

[10] C. Zhou, L.-T. Chia, and B.-S. Lee, "DAML-QoS Ontology for Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS 2004)*, 2004, p. 472.

[11] V. X. Tran and H. Tsuji, "A Survey and Analysis on Semantics in QoS for Web Services," *Proceeding of the 23rd International Conference on Advanced Information Networking and Applications (AINA 2009)*, vol. 0, pp. 379–385, 2009.

[12] G. Pierra, "Context Representation in Domain Ontologies and its Use for Semantic Integration of Data," *Journal Of Data Semantics (JODS)*, vol. X, pp. 34–43, 2007.

[13] M. Dean and G. Schreiber, *OWL Web Ontology Language Reference*, World Wide Web Consortium, 2004, http://www.w3.org/TR/owl-ref.

[14] R. C. Boer, P. Lago, A. Telea, and H. V. Vliet, "Ontology-Driven Visualization of Architectural Design Decisions," in *Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA 2009)*.

[15] S. Jean, Y. Aït-Ameur, and G. Pierra, "Querying Ontology Based Database Using OntoQL (an Ontology Query Language)," in *Proceedings of On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences (ODBASE 2006)*, vol. 4275. Springer, 2006, pp. 704–721.

[16] P. Kruchten, "An ontology of architectural design decisions in software intensive systems," in *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, October 2004, pp. 54–61.

[17] W. Siberski, J. Z. Pan, and U. Thaden, "Querying the semantic web with preferences," in *In Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, 2006, pp. 612–624.

[18] P. Gurský, T.Horváth, J. Jirásek, and P. Vojtás, "User preference web search – experiments with a system connecting web and user," in *To appear in the Computing and Informatics Journal*, 2008, pp. 25–32.

[19] A. Toninelli, A. Corradi, and R. Montanari, "Semantic-based discovery to support mobile context-aware service access." *Computer Communications*, vol. 31, no. 5, pp. 935–949, 2008.

[20] D. Tapucu, S. Jean, Y. Aït-Ameur, and M. O. Ünalir, "An extention of ontology based databases to handle preferences," in *Proceedings of 11th International Conference on Enterprise Information Systems (ICEIS 2009)*, 2009, pp. 208–214.

[21] G. Pierra, H. Dehainsala, Y. Aït-Ameur, and L. Bellatreche, "Base de Données à Base Ontologique : principes et mise en œuvre," *Ingénierie des Systèmes d'Information*, vol. 10, no. 2, pp. 91–115, 2005.

[22] H. Dehainsala, G. Pierra, and L. Bellatreche, "OntoDB: An Ontology-Based Database for Data Intensive Applications," in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, vol. 4443. Springer, 2007, pp. 497–508.

[23] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A QoS-Aware Selection Model for Semantic Web Services," in *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC 2006)*, 2006, pp. 390–401.