DAML-QoS Ontology for Web Services

Chen Zhou, Liang-Tien Chia, Bu-Sung Lee
Center for Multimedia & Network Technology
School of Computer Engineering, Nanyang Technological University
Email: {pg04878518, asltchia, ebslee}@ntu.edu.sg

Abstract

As more and more Web services are deployed, Web service's discovery mechanisms become essential. Similar services can have quite different QoS levels. For service selection and management purpose, it is necessary to explicitly, precisely, and unambiguously specify various constraints and QoS metrics for Web services descriptions. This paper provides a novel DAML-QoS ontology as a complement for DAML-S ontology to provide a better QoS metrics model. Three layers are defined together with clear role descriptions for developments. Cardinality constraints are utilized to describe the QoS property constraints. Basic profile is presented for general web service's description and the speed startup of ontology definition. Matchmaking algorithm for QoS property constraints is presented and different matching degrees are described. When incorporated with DAML-S, multiple service levels can be described through attaching multiple QoS profiles to one service profile. Well-defined Metrics can be further utilized by measurement organizations to guarantee the promised service level.

keywords*: Ontology, Web Services Discovery, Matchmaking, QoS*

1. Introduction

With the industry's efforts in promoting the use of web services, a huge number of web services are being developed and made available on the web. Service requesters are presented with a group choice of service offers that provide similar services. Different offers may have quite different quality of service. This will require more sophisticated patterns of service discovery and negotiation. For service selection and management purpose, it is necessary to explicitly, precisely, and unambiguously specify various constraints, QoS metrics, service level objectives, and other contracts between Web Services. The formal specification of constraints and SLAs and the differentiation of service have been researched extensively in computing and telecommunications. However, web services are XML-based protocol stacks and have its own specific features. The former researches could not be applied directly to Web Services. Furthermore, Web Service discovery, composition, and cooperation need to be more dynamic, automatic and across enterprise boundaries.

The semantic web can be a promising solution. It requires that data be not only machine readable, but also machine understandable. With the help of Semantic Web, application developers should not worry about how to interpret the information found on the Web, as ontologies will be used to provide vocabulary with explicitly defined and machine understandable meaning. One common ontology for Web Services which has been designed for the purpose of describing web services, is the DAML-S ontology[5]. DAML-S aims to make Web Services computer-interpretable and to enable automated Web service discovery, invocation, composition and monitoring. However, the specification has not provided a detailed set of classes, properties and constraints to represent QoS descriptions. We have tried to develop a proper QoS ontology design pattern for the formal specification of various types of constraints and QoS metrics as a complement to the DAML-S. This novel QoS ontology is based on DAML+OIL and named DAML-QoS. It has unique characteristics with regard to the semantic technologies: better machine understandability, interoperability, unambiguousness and extensibility. The corresponding matchmaking algorithm is also presented. As a following step of the DAML-S service profile's matchmaking, our work facilitates the QoS selection between similar semantic service advertisements. The metric ontologies may also provide a powerful solution for measurement organization to monitor and bill against the agreed upon SLAs.

This paper is organized as follows: In this section, we have introduced the motivation for our research. Section 2 introduces the background information of the description logic and DAML-S. In Section 3, we give the modeling definition of the QoS ontology and its relationship with DAML-S. Section 4 describes the matchmaking algorithm for our QoS ontology. In Section 5 we review some related works and compare our approach to these works. At the end, we

summarize the conclusions and future work.

2. Background

In this section we will give an overview of Semantic Web languages and Web Services description efforts.

2.1. Ontology Languages

Ontology plays a key role in the Semantic Web by providing machine readable vocabularies used by applications to understand the shared meanings. DAML+OIL[4] is an ontology language that has been designed specifically to be used in the Semantic Web and it is based on RDF and RDF Schema. Its well-defined semantics is similar to the description logic $SHIQ(D)$.DAML+OIL describes the structure of a domain in terms of classes and properties. Like $SHIQ(D)$, DAML+OIL (March 2001) also supports the use of datatypes in class description. By defining the service descriptions upon the semantics provided by DAML+OIL, we can utilize a DL reasoner to make inference and classify descriptions written in DAML+OIL.

2.2. Description Logic

The description logic is based on the notion of concepts (classes) and roles (binary relations), and is mainly characterized by constructors that allow complex concepts and roles to be built from atomic ones[7]. A DL reasoner can check whether two concepts subsume each other. In DL, an interpretation $I = (\triangle^I; \cdot^I)$ consists of a set $\triangle^I$, called the domain of I, and a valuation $\cdot^I$ which maps every concept to a subset of $\triangle^I$ and every role to a subset of $\triangle^I \times \triangle^I$ such that, for all concepts, roles, and nonnegative integers, the properties are satisfied.

The basic DL S does not fulfill our requirement of reasoning with cardinality restrictions on roles and datatypes. We therefore use the DL $SHIQ(D)$ to reason with DAML+OIL descriptions, which include, e.g., cardinality restrictions on roles, and datatypes. A more detailed discussion of DLs is out of the scope of this paper, which can be found in [1].

2.3. DAML-S

DAML-S[5] is a DAML+OIL ontology for describing Web Services. Through the tight connection with DAML+OIL, DAML-S aims to make Web Services computer-interpretable and to enable automated Web service discovery, invocation, composition and monitoring. It defines the notions of a Service Profile (what the service does), a Service Model (how the service works) and a Service Grounding (how to use the service). As a DAML+OIL ontology, DAML-S retains all the benefits of Web content described in DAML+OIL. It enables the definition of a Web services vocabulary in terms of objects and the complex relationships between them, including class, subclass relations, cardinality restrictions, etc.[4] It also includes the XML datatype information.

However, Cardoso et. al.[3] point out that significant improvement for the QoS model should be made to supply a realistic solution to DAML-S' users. One current limitation of DAML-S' QoS model is that it does not provide a detailed set of classes and properties to represent quality of service metrics. The QoS model needs to be extended to allow a precise characterization of each dimension. This is the motivation of our current work for the QoS ontology.

3. Modeling

Web Services QoS ontology, especially for service discovery purpose, is the focus of our work. Here we design a web services domain-specific QoS ontology in order to achieve an agreement at the semantic level between various parties. Our ontology contains three layers: the QoS profile layer designed for matchmaking purpose; the QoS property definition layer for elaborating the property's domain and range constraints; the metrics layer for metrics definition and measurement. In our prototype, we define the ontology in DAML+OIL. For the purpose of clarity and compactness, in this paper we will use the DL notions in place of the DAML+OIL syntax for the T-Box definition.

3.1. QoS Profile Layer

In the QoS profile layer, we define QoSProfile as a common superclass for QoS matchmaking concept ProviderQoS, InquiryQoS and TemplateQoS. They can be formulated as:

$$\begin{aligned} QoSProfile &\sqsubseteq \top \\ ProviderQoS &\sqsubseteq QoSProfile \\ InquiryQoS &\sqsubseteq QoSProfile \\ TemplateQoS &\sqsubseteq QoSProfile \end{aligned}$$

The QoSProfile is logically used for three roles. ProviderQoS is the advertisement ontology published by the service provider. InquiryQoS is the service requester's inquiry ontology for QoS matchmaking. The TemplateQoS is stored by any user for further usage or modification. The above definition, however, has not provided any constraints over QoSProfile so that the ProviderQoS contains all the possible QoS combinations for the published service. No constraint means no useful information for the QoS matchmaking process and this makes no sense. Our solution is to use property definition and cardinality to define the QoS constraints. Cardinality is chosen

instead of concrete datatype because of the following reasons:

- All the QoSProfile layer ontologies are described in the T-Box to make the description uniform and reduce the matchmaking component's complexity.
- Current DL reasoners normally have better support for the subsumption reasoning in the T-Box than concrete datatype reasoning in A-Box. Through classification, taxonomy is built up by the reasoner's predefined and tested algorithm.
- Cardinality is constrained over property which has its own domain and range constraints. These constraints make the matchmaking more specific and precise.

This solution can cause a potential problem. The cardinality ranges over nonnegative integers only so that the cardinality constraints cannot represent the real number value. This imprecise will cause a maximum error that can reach one half metric unit. However, by proper selection of metric unit, the error can be constrained within one half metric unit and the metric unit can be selected according to the precision requirement. Furthermore, most measurements by the observer are within the nonnegative integer domain. Observers normally obtain the current resource size or count for certain events. Such information's representation in observer's internal data structures is normally nonnegative integers, such as response time, bit rate, etc. Even if the observed data cannot be precisely described by integer domain, through the using of more fine-grained metric unit, this problem can always be solved. For example, using dollar as the cost unit is not a good solution but we can change the unit into cent to ensure the correctness. If different QoS metric units for the same property are used in different advertisement, ontology translation is needed to normalise the metric units.

In the normal web service development process, the service provider hosts the web services and publishes the service description information to the UDDI[2]. It is the service provider's task to setup this layer and provide enough and correct property and cardinality constraint information for service requester to discover and locate the proper service. The cardinality can be viewed as abstract resource tokens to represent the resources. The matchmaking process will be discussed in Section 4.

3.2. QoS Property Definition Layer

In addition to the property name, QoS property definition constrains the property's domain and range information. The domain is the class which has the special QoS property. We specify five QoSProfile classes for the QoS property's domain: QoSCore, QoSinput, QoSOutput, QoSPrecondition and QoSEffect, and they can be expressed as:

$$
\begin{aligned}
QoSProfile &\sqsubseteq \top \\
QoSCore &\sqsubseteq QoSProfile \\
QoSInput &\sqsubseteq QoSProfile \\
QoSOutput &\sqsubseteq QoSProfile \\
QoSPrecondition &\sqsubseteq QoSProfile \\
QoSEffect &\sqsubseteq QoSProfile
\end{aligned}
$$

The QoSCore stands for the normal QoS property's origination. This is the default QoS property's domain class. From the service requester's view, if there's no difference in QoS properties based on input and output, this property's domain is assumed to be set on the QoSCore. Otherwise, if two QoS properties are not the same on the input and output, their domains are set as QoSInput and QoSOutput respectively. For example, to a format covert service, we can have two QoS properties: the input bit rate and the output bit rate. These two values can be different. Thus we can set the inputBitRate property's domain to QoSInput and the outputBitRate property's domain to QoSOutput. Furthermore, since a service may require external conditions to be satisfied to ensure that it can provide the promised QoS level, and it may have the effect of changing the QoS condition, the QoSProfile ontology describes the QoSPrecondition required by the service and the expected QoSEffect that result from the execution of the service. For example, some computational service can require that the throughput of the request is within 50 times per minute so that it can guarantee the published QoS level. Such property's domain is defined as QoSPrecondition. After the execution, the service has the effect of lower throughput because the machine needs to be cooled down. This property's domain is defined in QoSEffect.

The range of the QoS property is defined within the QoS metric class. The QoS metric classes are defined in the QoS Metrics Layer (See Section 3.3). With the range constraints together with the domain constraints, the QoS properties are precisely specified. After the definition of the QoS properties, cardinality constraints are ready to be added on these defined properties in QoS Profile Layer (See Section 3.1) for matchmaking purpose.

The development of QoS ontology allows a new role of QoS designer who designs customized QoS properties, and selects or invents the suitable QoS metrics for the QoS properties. It is the service QoS designer's and web services vendor's task to define the available property types, their domain constraints and range constraints. Newly invented QoS metrics are put into the QoS metrics hierarchy taxonomy while the metrics' individual definition in A-Box is left to the QoS measurement organization. The proper definition of this layer is the key for QoS profile definition and matchmaking.

3.3. QoS Metrics Layer

The definition of this layer has two purposes: firstly, this layer defines proper QoS metrics for the QoS property's range definition. Secondly, this layer defines precise semantic meanings for service measurement organization to measure the service and check against the guarantee.

The service QoS metrics are divided into AtomicMetrics and ComplexMetrics, which are showed as follows:

$$\begin{aligned} Metric &\sqsubseteq \top \\ AtomicMetric &\sqsubseteq Metric \\ ComplexMetric &\sqsubseteq Metric \end{aligned}$$

The Metric class is a common superclass for all metrics. The Metric class has related properties *unit*, *value*, *metricName*. The domains of these properties are indicated as the Metric class. Their ranges are *Unit*, *&xsd;#nonNegativeInteger*, and *&xsd;#string* respectively. Unit class indicates the metric value's unit. The Unit has its own taxonomy hierarchy. The *&xsd;* is the entity macro standing for the XML Schema namespace. The value property has the nonNegativeInteger as its range, which conforms to the property cardinality constraints in QoS Profile Layer. The value in the metric class is within individual declaration and it is used for web services partners to check against the published cardinality constraints in QoS Profile Layer, rather than matchmaking. By proper selection of the unit, the nonNegativeInteger is a practical choice for QoS value as discussed in Section 3.1.

The atomic metrics are directly measured by the observer. They contains child metrics and the measurement result is retrieved from the observer directly. That is, the measurement organization gets the access point from the AtomicMetric individual, invokes the observer's retrieval service and then extracts the QoS data from the observer. Therefore, for each AtomicMetric there must be an observer's access point that enables the measurement organization to get the QoS data.

The complex metrics are composed of other (AtomicMetric or ComplexMetric) metrics. The operands property in ComplexMetric points to these child metrics (AtomicMetrics or ComplexMetric). The function property in ComplexMetric points to the Function class which describes how to process the operand metrics. Through the ComplexMetric, the high level metrics can be mapped to the lower level metrics in clear semantics. The metric aspect can also be described through ComplexMetric, such as percentile, mean, variance, and frequency.

Each QoS metric is a subclass of the AtomicMetric or ComplexMetric. The metrics' taxonomy is designed by QoS designer or web service vendor in the T-Box. The individual of each AtomicMetric and ComplexMetric is defined by measurement organization in A-Box. Because each metric class can has multiple individuals in the A-Box, different measurement organizations can offer multiple choices for the observers and complex metrics. The service provider and service requester can choose the proper metric individuals for the QoS monitoring and supervision. The proper definition of this layer is the key for QoS monitoring.

3.4. Basic Profile

Anbazhagan et. al. [10] highlighted that the major requirements for supporting QoS in Web Services include Performance, Reliability, Security, etc. [14] divided the services QoS into three categories: system centric, process centric and information centric. The performance, reliability, security, etc. are located in the system centric category. These general QoS metrics are normally needed in QoS description. To facilitate the speed startup in using the QoS ontology, we design a basic profile according to system centric QoS category.

The basic profile contains response time, cost, reliability and throughput.

- Response Time is defined as the total time needed by the service requester to invoke the service. It is measured from the time the requester initiates the invocation to the time the requester receives the last byte of the response.
- Cost represents the cost associated with the execution of the service. It is necessary to estimate the guarantee that financial plans are followed. The cost can be broken into major components, which include the service execution cost and network transportation cost.
- Reliability corresponds to the likelihood that the service will perform when the user demands it and it is a function of the failure rate. Each service has two distinct terminating states: one indicates that a web service has failed or aborted; the other indicates that it is successful or committed. By appropriately designed redundancy, one can build highly reliable systems from less reliable components.
- Throughput represents the number of Web service requests served at a given time period. It is the rate at which a service can process requests.

Using the basic profile, service provider and service requester can easily write the QoS descriptions for a general web service. After the setting of the cardinality constraints, service provider completes the QoS Profile Layer's definition. Figure 1 shows an example of QoS advertisement only. Suppose that we want to specify the QoS level of a service in the QoS Profile Layer, in which the response time is no more than 20 seconds and the cost is no more than 1 dollar. In DL syntax, this advertisement can be written as:

$$Advert \doteq QoSProfile \sqcap (\leq 100 cost.CostUSCentMetric) \sqcap (\leq 20000 responseTime.RespMSMetric)$$

Similar to the advertisement, service requesters can define an inquiry in which the response time of the service should be no more than 40 seconds and the cost should be no more than 5 dollars. This inquiry can be expressed as:

$$Inquiry \doteq QoSProfile \sqcap (\leq 500 cost.CostUSCentMetric) \sqcap (\leq 40000 responseTime.RespMSMetric)$$

The template can be defined in the similar way. Once defined, the template can be reused by service providers or service requesters through adding additional constraints over the template.

3.5. Extensibility

Each web service may have different QoS metrics to evaluate and describe its QoS information. The basic profile provides a speed startup for general web services, however some other services have their own specific QoS properties. This requires that the QoS description for service discovery should have good extensibility to meet different user's demands. One of the semantic web's design principles is to allow anyone to comment about anything. This design principle meets the extensibility requirement quite well. By using DAML+OIL ontologies, extensibility is naturally achieved.

The different scenarios for the creation and maintainance of the QoS ontology are as follows:

- *Green field*: In this scenario, the developer starts completely from scratch, creating all the layer's ontologies. With this approach, QoS designer first designs the customized QoS property, and then selects or defines the QoS metrics for the property's range constraint. The new QoS metrics are put in the metric taxonomy. The QoS metric's individuals are then defined by measurement organizations. The cardinality constraints for the QoS properties are set by the service provider.
- *Bottom up*: The *bottom up* scenario follows along the same lines as the *green field*, with the exception that the QoS metrics and properties have already been defined by the QoS designer or by the web services vender. The provided basic profile is located in this category. Remaining thing is to setup the cardinality constraints for the properties by the service provider.
- *Top down*: The *top down* scenario is a bit different. The property domain and constraints have already been defined while the QoS metrics are considered but not properly defined or the original QoS metrics are not suitable. The QoS designer or service vendor will select one metric in the taxonomy or define a new metric. The newly defined metric's individuals will be defined by the measurement organization.

When the service provider has built up a common QoS Profile for its specific service domain, this common QoS Profile can be defined as a template, which is a stong assistant for extensibility. Based on the template, service provider can expend the QoS Profile Layer, add more QoS properties and set stricter constraints over properties. Using the template to inherit the original QoS Profile avoids building the whole profile block from scratches. For example, suppose that we've made a basic profile template named BPTemplate. Based on BPTemplate, we're going to define two storage services: one provides 10MB space for the service requester and the other provides 100MB space. This can be described as follows:

$$StorageAdvert1 \doteq BPTemplate \sqcap (\leq 10 storage.storageMBMetric)$$
$$StorageAdvert2 \doteq BPTemplate \sqcap (\leq 100 storage.storageMBMetric)$$

This ontology inheritance ability helps to achieve easy extensibility. Meanwhile, the inheritance indicates more specified and constrained QoS descriptions. Inheritance cannot achieve fewer constraints on QoS properties. For example, if the original template's response time constraint is ≤ 10000, and the inherited QoS class want to set the response time constraint as ≤ 20000, then this will not take effect. The inherited class still has the response time constraint as ≤ 1000. This will be discussed further in Section 4. Therefore, the template's inheritance should be carefully designed so that the subclasses of the templates are of stricter constraints.

3.6. Relationship with DAML-S

The approach described here allows a service developer to take advantage of the complementary strength of both DAML-S and our QoS ontology design model. On one hand (service profile side), service developer benefits by making use of DAML-S' service profile model for semantic matchmaking of service descriptions, as well as the well defined process model and the grounding information. On the other hand (QoS profile side), the developer benefits from the use of DAML-QoS' QoS prfile model for QoS matchmaking, as well as the QoS metric layer's definition for the QoS measurement.

To connect the DAML-S with our QoS ontology, the hasServiceProfile property with range constrain ServiceProfile is required to be added in the QoS Profile Advertisement. Multiple QoS Profiles of one Web Service can refer to the same service profile, and they have different constraints

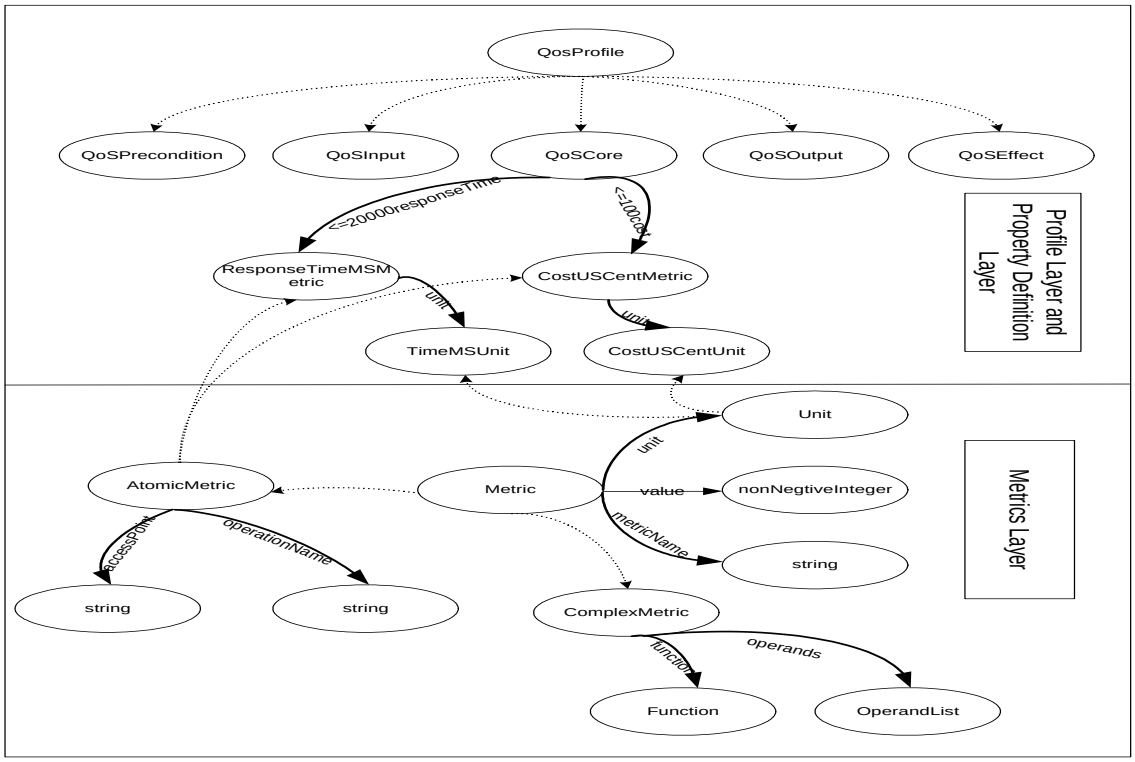

Figure 1. Advertisement Ontology Layers Example

over the contained QoS property constraints which are in a lower level than the QoS Profile. This provides multiple service level objectives (refers to the different QoS Profiles) to the service requester, each with different capability, performance, price, etc. The service requester can choose the most suitable one according to their customized inquiry. Dynamic adaptation is possible based on multiple QoS Profiles.

With the help of the Basic Profile and DAML-S' process ontology, algorithms can be implemented for the automatic computation of QoS metrics for processes based on atomic tasks and sub-processes' QoS metrics. The analytic model used in [3] is a good example.

4. Matchmaking

Matchmaking here is defined as a process that requires a repository to take an inquire as input, and to return all the published advertisements which may potentially satisfy the QoS requirements specified in the input inquiry.

4.1. Constraint Order Definition

The novel of QoS matchmaking lies in the matching of different QoS advertisements with different QoS constraints. We must determine whether the guarantee given in the provider's advertisement is no looser than the one inquired by the service requester is true.

Cardinality constraints for the property contain $\geq$ and $\leq$ operators. These operators on the nonnegative integer domain are totally ordered sets. To the same property, what is better QoS in the description is defined as follows: To $\geq$ operator, $\geq m$ means better QoS than $\geq n$ if $m > n$. To $\leq$ operator, $\leq m$ means better QoS than $\leq n$ if $m < n$. If some metric does not have the nonnegative integer values as its domain, mapping is required for the metric's definition. Take the score degree metric as an example: there exist five levels: A, B, C, D and F. A definition for these levels on the nonnegative integer domain is required for the metric ontology's definition. One solution is to define 5, 4, 3, 2, 1 for A, B, C, D, F respectively. Then all those who pass the exam can be described as ≥ 2.

Better QoS profile's description will subsume the worse one. Let's assume A and B are better and worse advertisements respectively. There exist two cases:

Firstly, A and B has the same list of l properties: $P_1.C_1, P_2.C_2, ..., P_l.C_l$, in which P_i is property names and C_i is range class names. Without loss of generality, we assume that the $\leq$ cardinality constraint is used in each case. We have definition of A and B here:

$$A \doteq (\leq k_1 P_1.C_1) \sqcap (\leq k_2 P_2.C_2) \sqcap ... \sqcap (\leq k_l P_l.C_l)$$
$$B \doteq (\leq n_1 P_1.C_1) \sqcap (\leq n_2 P_2.C_2) \sqcap ... \sqcap (\leq n_l P_l.C_l)$$

in which $k_i > n_i$ (A has better QoS description), $i = 1, 2, ..., l$. To each property we have $(\leq k_i P_i.C_i) \sqsubseteq (\leq n_i P_i.C_i)$. By conjunction on these properties, we have $A \sqsubseteq B$.

On the contrary, if $A \sqsubseteq B$ and they have same property list, A will have the better QoS profile description than B. If this is not true, there's some property $(\leq k_t P_t.C_t)$ in A and $(\leq n_t P_t.C_t)$ in B in which $n_t > k_t$ hence $(\leq n_t P_t.C_t) \sqsubseteq (\leq k_t P_t.C_t)$. Because the property $P_t.C_t$ is not defined in the A-Box, we can define the property in proper manner and make the individual x to satisfy that $x \in \triangle^I$, $x \in (\leq k_t P_t.C_t)^I$, $x \notin (\leq n_t P_t.C_t)^I$ and $x \in (\leq k_i P_i.C_i)^I$ where $i = 1, 2, ..., l$ and $i \neq t$. Therefore $x \in A^I$ while $x \notin B^I$, this contradicts the premise $A \sqsubseteq B$. Therefore, $n_t <= k_t$ and A has the better QoS profile description than B.

Secondly, A has all the B's QoS properties as well as some additional properties. All the same properties in A are better than B and we define all these same properties' conjunction in A as class AS. Then we have $A \sqsubseteq AS \sqsubseteq B$ and vice versa, which can be proved in the similar way to the previous case.

4.2. Matchmaking Algorithm

Formally the matchmaking can be specified as: For a given inquiry P, the matchmaking algorithm should return the set of all the published advertisements which are compatible. Two QoS ontology descriptions, say C1 and C2, are compatible iff their intersection is satisfiable:

$$compatible(C1, C2) \Leftrightarrow \neg(C1 \sqcap C2 \sqsubseteq \perp)$$

All the compatible advertisements will be added to the result set. However, we need to introduce the definition of the degree of match to distinguish different advertisements. The matching degree definition in our algorithm is different from [11, 8]. Some of the definition are as follows:

- **Subsume** If request R is super-concept of advertisement A,we call the match Subsume; that is, $A \sqsubseteq R$.
- **Exact** If advertisement R and request A are equivalent concepts, this is called the match Exact; that is, $A \equiv B$.
- **PlugIn** If request R is sub-concept of advertisement A, we call this match PlugIn; that is, $R \sqsubseteq A$.
- **Intersection** If the intersection of advertisement A and request R is satisfiable, we call the match Intersection; that is, $\neg(A \sqcap R \sqsubseteq \perp)$
- **Disjoint** Otherwise it is disjoint; that is, $A \sqcap B \sqsubseteq \perp$.

Degrees of the match are organized in a discrete scale. Subsume matches are considered the preferable match, since we can expect that the advertisement with better QoS description will subsume the inquiry description; Exact matches are the next best, since the advertisement is exactly the same to the requirement's description; PlugIn matches are considered to be the third best, since the advertisement does not fully provide the required QoS level according to the inquiry; Intersection is supposed to be the fourth best, since it means that the advertisement is not incompatible with the inquiry; and Disjoint is the lowest level since it shows that nothing could satisfy both the advertisement and the inquiry, which is a failed match. Intersection matches are not necessarily of worse QoS than PlugIn matches.

With the definition of match degrees, we can use a DL reasoning engine to match a request. We use the RACER system to compute a QoSProfile hierarchy for all advertised services. When an inquiry arrives, RACER is used to classify the requester's QoSProfile R, that is, to compute $R's$ subsumption relationships against all the advertisement QoSProfiles. Advertisements with QoSProfiles subsumes but not equal to R are considered to be Subsume matches. Those with QoSProfiles equivalent to R are considered as Exact matches, and those with QoSProfiles subsuming but not equal to R are considered to be Plugin matches. Then RACER is used to classify $\neg R$. Advertisements with QoSProfiles subsuming but not equal to $\neg R$ are considered to be Intersection matches, and those subsumed by $\neg R$ are considered to be Disjoint matches. For example, the $Advert$ and $Inquiry$ defined in Section 3.4 satisfies that $Advert \sqsubseteq Inquiry$ so that the $Advert$ will be included in the result set.

5. Related Works

There're many research works that target the describing, advertising and signing up to Web and Grid services at defined QoS levels. This includes HP's Web Services Management Language(WSML) and framework[12], IBM's Web Service Level Agreement (WSLA) language[9], the Web Services Offer Language (WSOL)[13] as well as approaches based on WS-Policy[6].

WSML and WSLA were developed for the XML-based specification of custom-made SLAs for Web Services. Their SLAs contain QoS constraints, prices and management information. They're oriented towards management applications in enterprise scenarios and they are accompanied by appropriate management infrastructures. Some support for templates is available in WSML and WSLA. WSOL provides formal representation of various constrains as well as management statements. Its major feature is richer set of reusability constructs and lightweight management infras-

tructure. The definition of QoS metrics about how they are measured or computed is done in external ontologies. WS-Policy is a general framework for the specification of policies for Web Services. The details of the specification for particular categories of policies will be defined in specialized languages. It is flexible and extensible because policies are not limited in certain places and its specification is extensible through additional specifications. However, when the new specification will appear and how the polices are monitored and evaluated remain a problem.

Our QoS Profile is based on DAML+OIL layer instead of pure XML layer. The advertisement is specified in a more unambiguous manner because of the stricter constraints over the cardinality, domain and range. The published web services should not need human intervention to understand the meaning of SLAs, and to monitor and guarantee their compliance. In addition, unification and standardization of the well defined cardinality constraints and metric semantics will reduce the programming effort of supporting framework and achieve better code reuse. The extendibility and openness of ontology facilitates the share of experiences and fastens the development cycle. The well-established reasoning tool is also a great help to check the consistency between QoS metrics and ensures a quick development for the matchmaking frameworks. However, we mainly focus on using this ontology as descriptive advertisement for the service discovery purpose instead of SLA assignment, monitoring, billing and analysis purpose in the web service life cycle. WSLA and some other works made good efforts on the SLA supporting framework for web services. The SLA supporting system based on ontology layer can be of better automaticity but it remains a research issue.

6. Conclusions and Future Work

This paper provides a novel DAML-QoS ontology as a complement for the DAML-S ontology to provide a better QoS metrics model. It is mainly designed for the matchmaking purpose. The well-defined Metric can be further utilized by measurement organizations to guarantee the SLAs for the service. The ontology contains three definition layers: the QoS Profile Layer, the QoS Property Definition Layer and QoS Metrics Layer. The roles for the development of each layer are described. A basic profile is recommended for normal web services' usage. Additional specific properties can be added above the basic profiles for certain service categories, such as the storage service, computational service, etc. The matchmaking algorithm is presented for the descriptions. One service profile can also have multiple QoS profiles for the choice of service level objectives. The recent related works, such as WSLA, WSML, etc., are mainly based on XML layer so that they do not have the semantic web technology's advantages such as good machine understandability, interoperability, unambiguousness, and better extensibility.

The QoS matchmaking is a following step of the service profile matchmaking. As part of our future work, we would incorporate the QoS matchmaking and service profile matchmaking into our current QoS-Aware service discovery framework[15]. Furthermore, the metric class can be further studied to provide better management control together with supporting frameworks.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] T. Bellwood, L. Clmen, and C. von Riegen. UDDI Version 3.0.1 Specification, Nov 2003.

[3] J. Cardoso, A. Sheth, and J. Miller. Workflow Quality of Service. In *Proceedings of the International Conf. on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference*, 2002.

[4] DAML+OIL. The DAML+OIL Language. 2001.

[5] A. A. et al. DAML-S: Web Service Description for the Semantic Web. In *Proc. Int'l Semantic Web Conf. (ISWC 02).*, 2002.

[6] M. Hondo and C. Kaler. Web Services Policy Framework (WS- Policy) Version 1.0, December 2002.

[7] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, pages 161–180, 1999.

[8] L. Li and I. Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology. In *the International World Wide Web Conference (WWW2003)*, 2003.

[9] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification, v1.0, Jan 2003.

[10] A. Mani and A. Nagarajan. Understanding quality of service for Web services. 2002.

[11] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf.*

[12] A. Sahai, A. Durante, and V. Machiraju. Towards Automated SLA Management for Web Services. HPL-2001-310 (R.1), 2002.

[13] V. Tosic, B. Pagurek, and K. Patel. WSOL - A Language for the Formal Specification of Classes of Service for Web Service. In *the Int. Conf. on Web Services (ICWS03)*, 2003.

[14] G. Weikum. Towards guaranteed quality and dependability of information service. In *8th GI Fachtagung: Datenbanksysteme in Buero, Technik und Wissenschaft*, 1999.

[15] C. Zhou, L.-T. Chia, B. Silverajan, and B.-S. Lee. UX—An Architecture Providing QoS-Aware and Federated Support for UDDI. In *the Int. Conf. on Web Services (ICWS03)*.