Web Services Discovery on DAML-Qos Ontology

Chen Zhou, Nanyang Technological University, Singapore
Liang-Tien Chia, Nanyang Technological University, Singapore
Bu-Sung Lee, Nanyang Technological University, Singapore

ABSTRACT

As more and more Web services are deployed, Web service's discovery mechanisms become essential. Similar services from different sources often exhibit quite different Quality of Service (QoS) levels. For service selection and management purpose, it is necessary to explicitly, precisely, and flexibly specify various constraints and QoS metrics for Web services descriptions. This chapter provides a novel ontology as a complement to DARPA Agent Markup Language—Service (DAML-S) Ontology to provide a better QoS metrics model. Three layers are defined, together with clear role descriptions for development. Cardinality constraints are utilized to describe the QoS property constraints. A new matchmaking algorithm for QoS property constraints with multiple matching degrees is proposed. When incorporated with DAML-S, the multiple service level objectives can be described by assigning multiple QoS profiles to one service profile. A prototype system was developed, which demonstrates the feasibility for small to medium-sized service advertisement repository.

Keywords: matchmaking; ontology; QoS; Web services discovery

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 qualities of service. This will require more sophisticated patterns of service discovery and negotiation.

For service selection and management purposes, it is necessary to explicitly, precisely, and flexibly specify various constraints, QoS metrics, service level objectives, and other contracts between Web services. A Service Level Agreement is a contract between the provider and the user that specifies the level of service expected during its term. The formal specifications of constraints and Service Level Agreement (SLA) have been researched extensively in computing and telecommunications. However, Web services are Exten-

sible Markup Language- (XML-) based protocol stacks with their own specific features. Thus, previous researches in this area could not be applied directly to Web services. Furthermore, Web service discovery, composition, and cooperation are more dynamic and automatic, and cross enterprise boundaries.

The semantic Web technology is a promising solution. It requires that data not only be 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. DAML-S Ontology (DAML-S Coalition, 2002) is a well-known ontology that has been designed for the purpose of describing Web services. 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 developed a more comprehensive 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 DARPA Agent Markup Language—Quality of Service (DAML-QoS). It has unique characteristics with regard to the semantic technologies: better machine understandability, interoperability, unambiguousness, and extensibility. The corresponding matchmaking algorithm is presented. In addition to the DAML-S service profile's matchmaking, our work facilitates the QoS selection between semantically similar services. The metric ontologies also can provide a powerful solution for measurement organization to monitor and bill against the agreed upon SLAs.

This chapter is organized as follows: In this section, we have introduced the motivation for our research. This is followed by an introduction on the background information of the description logic and DAML-S. In the next section, we review some related works and compare our approach to these works. Then we give the modeling definition of the QoS Ontology and its relationship with DAML-S. Further, we describe the matchmaking algorithm for our QoS ontology. After that, the prototype system design and experiment results are discussed. Finally, we summarize the results and discuss about future work.

BACKGROUND

In this section we will give an introduction to Semantic Web and Web services description languages.

Ontology Languages

Ontology plays a key role in the Semantic Web by providing machine-readable vocabularies for applications to understand the shared meanings. DAML+OIL (DAML+OIL, 2001) is the successor of Ontology Inference Layer (OIL), defined in collaboration with research groups from the DARPA-sponsored DAML program, following the original versions of OIL. It is an ontology language that has been designed specifically to be used in the Semantic Web, and it is based on Resource Description Framework (RDF) (Beckett & McBride, 2004) and RDF Schema. Its well-defined semantics is similar to the description logic *SHIQ(D)* (Horrocks & Sattler, 2001). DAML+OIL describes the structure of a domain in terms of classes and

properties. Like *SHIQ(D)*, DAML+OIL 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 Description Logic reasoner to make inference and classify descriptions written in DAML+OIL. The latest Semantic Web's Ontology Language (OWL) (W3C Committee, 2004) has evolved to be a new promising ontology language to substitute the current place of DAML+OIL. We still build up our ontology based on DAML+OIL because of its better tool support. There is not much difficulty migrating the current system to OWL in the future.

Description Logic

Description logics (DLs) are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies. 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 (Horrocks et al., 1999). A DL reasoner can check whether two concepts subsume each other. Table 1 shows the semantics of description logic *SHIQ*. In DL, an interpretation $I = (\Delta^I, \cdot^I)$ consists of a set Δ^I, called the domain of I, and a valuation $\cdot^I$ which maps every concept to a subset of Δ^I and every role to a subset of $\Delta^I \times \Delta^I$ such that, for all concepts, roles, and nonnegative integers, the properties in Table 1 are satisfied. $\#M$ denotes the cardinality of a set M, and $(R^I)^+$ denotes the transitive closure of R^I (Horrocks et al., 1999).

The basic DL S does not fulfill our requirement of reasoning with cardinality restrictions on roles and datatypes. Therefore, we use the DL *SHIQ(D)* to reason with DAML+OIL descriptions, which includes cardinality restrictions on roles and datatypes. A more detailed discussion of DLs, which can be found in Baader et al. (2003), is out of the scope of this chapter.

Table 1. Semantics of description logic SHIQ

Construct Name	Syntax	Semantics	
atomic concept	A	$A^I \subseteq \triangle^I$	
universal concept	$\top$	$\top^I = \triangle^I$	
atomic role	R	$R^I \in \triangle^I \times \triangle^I$	
transitive role	$R \in \mathbf{R}$	$R^I = (R^I)^+$	
conjunction	$C \sqcap D$	$C^I \cap D^I$	S
disjunction	$C \sqcup D$	$C^I \cup D^I$	
negation	$\neg C$	$\triangle^I \backslash C^I$	
exists restriction	$\exists R.C$	$\{x \mid \exists y\ y.\langle x,y\rangle \in R^I \text{ and } y \in C^I\}$	
value restriction	$\forall R.C$	$\{x \mid \forall y\ y.\langle x,y\rangle \in R^I \text{ implies } y \in C^I\}$	
role hierarchy	$R \sqsubseteq S$	$R^I \subseteq S^I$	H
inverse role	R^-	$\{\langle x,y\rangle \mid \langle y,x\rangle \in R^I$	I
number restrictions	$\geq nR$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^I\} \geq n\}$	N
	$\leq nR$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^I\} \leq n\}$	
qualifying number restrictions	$\geq nR.C$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^I \text{ and } y \in C^I\} \geq n\}$	Q
	$\leq nR.C$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^I \text{ and } y \in C^I\} \leq n\}$	

DAML-S

DAML-S (DAML-S Coalition, 2002) 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 classes, subclass relations, cardinality restrictions, and so forth (DAML+OIL, 2001). It also includes the XML datatype information. Currently, the DAML-S has evolved into OWL-S, which is based on OWL Ontology language (W3C Commitee, 2004).

DAML-S provides a good representation of a service's functional capability. However, Cardoso et al. (2002) pointed out that significant improvement for the QoS model should be made to supply a realistic solution to DAML-S' users. One 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 an important motivation of our current work for the QoS Ontology.

RELATED WORKS

Service discovery is normally a single process. However, this process can be divided into two phases:

Phase 1: Basic service requirements: This phase uses the capability aspect of the service, such as the service's functionality, input, output, precondition, and effect parameters. This phase ensures that the returned services meet the requester's basic requirement.

Phase 2: QoS matching: this phase identifies the most appropriate service for the current task by using the QoS information about each of the services returned from the first phase. This phase has become an important research area in process combination and resource discovery because dynamic Web service binding and invocation are preferred.

To realize the second phase, two efforts must be ready: the QoS description that describes the service's QoS information and the corresponding matchmaking algorithm. There are many research works that focus on describing and advertising Web services at defined QoS levels. These are normally application layer specifications that are hardware and platform independent. They include aspect-oriented approach (Jin & Nahrstedt, 2004) such as QuO framework (Zinky et al., 1997) and its QoS Description Language (QDL); object-oriented approach such as HP's QoS Modeling Language (QML) (Frolund & Koistinen, 1998); XML based QoS languages such as HP's Web services Management Language (WSML) and its framework (Sahai et al., 2002), IBM's Web Service Level Agreement (WSLA) language (Ludwig et al., 2003), and its supporting framework (Dan et al., 2004), the Web Services Offer Language (WSOL) (Tosic et al., 2003), as well as approaches based on WS-Policy (Hondo & Kaler, 2002).

QDL consists of three sublanguages, the contract description language (CDL), the structure description language (SDL), and the resource description language (RDL). The CDL specifies a QoS contract, which contains *nested regions* for a possible state of QoS; *transitions* for trigger behavior when region changes; *system condition objects* for measuring QoS information; and *callbacks* for notification. There is no special reusable construct for specification reuse.

QML (Frolund & Koistinen, 1998) is a non-XML-based specification for defining multi-category QoS specifications for components in distributed object systems. Through object-oriented approach, it provides specification refinement and simple contract types such as reliability and performance. Complex QoS specification also can be expressed, for example, using percentiles, variance, and frequency. Profile refinement and conformance are defined for profile management. QML supports the specification reusability through contract and profile refinement.

WSML (Sahai et al., 2002) and WSLA (Ludwig et al., 2003) were developed for the XML-based specification of custom-made SLAs for Web services. They define SLAs that contain QoS constraints, prices, and management information. In addition to the SLA definition, they are oriented toward management applications in enterprise scenarios. Appropriate management infrastructures are accompanied by these specifications. In the definition of SLA aspect, these specifications try to provide a precise and flexible solution. Some support for templates is available in WSML and WSLA to provide the flexibility.

WSOL (Tosic et al., 2003) provides formal representation of various constraints as well as management statements. Its major feature is its rich set of reusability constructs and lightweight management infrastructure. The definition of QoS metrics and how they are measured or computed is done in external Ontologies.

WS-Policy (Hondo & Kaler, 2002) 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 because policies are not limited in certain areas, 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 Ontology is based on DAML+OIL layer instead of pure XML layer. The advertisement is specified in a precise manner because of the strict constraints over the property's cardinality, domain, and range. To make an SLA machine readable, it is represented by one or more Service Level Objectives (SLOs). An SLO is normally a set of parameters and their values. The published Web services should not need human intervention to understand the meaning of SLOs and to monitor and guarantee the application's compliance. In our approach, each QoSProfile class represents an SLO definition. Interoperability is a big issue when there is a multiplicity of incompatible specifications. Using the Ontology approach, this is partially solved by partners agreeing on QoSProfile Ontology and QoS Metric Ontologies. A common Ontology (Fox & Gruninger, 1994) should be established for each domain so that all partners in the same domain speak and understand the same words. The extendibility and openness of Ontology facilitates the sharing of experiences and fastens the development cycle. In addition, unification and standardization of the well defined cardinality constraints and metric semantics will reduce the programming ef-

fort of supporting framework and achieve better code reuse.

Similar to the QML specification, we use the object-oriented approach to define our specification. The object-oriented approach is implicitly inherited from DAML+OIL to enable the specification's reusability. Any programmer who is familiar with the object-oriented design principle will have a relatively short learning curve for the specification design and reuse. Different from the QML's syntax, our approach is based on the XML syntax, and the semantic is based on DL logic. In this chapter, we focus mainly 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. Our Ontology design does not address the problem of what actions to trigger at runtime, if the QoS requirements cannot be satisfied. Therefore, some specification (i.e., QDL) is more expressive than DAML-QoS Ontology. WSLA and some other works made good efforts on the SLA supporting framework (Dan et al., 2004) for Web services. The SLA supporting system based on Ontology layer can provide better automaticity, but it remains a research issue.

As most of the previous QoS specifications define their own language syntax and semantics, a special engine is needed to proceed and understand the syntax and its semantics. The QML project mentioned that they want to create a mapping from QML to both Java and C++ so that they can represent QML specifications as Java objects at runtime (Frolund & Koistinen, 1998). However, the mapping process and the specification matchmaker are not well defined. For WSLA, a matchmaking algorithm is proposed to check the compatibility between two SLOs (Weilai Yang, 2003). This approach uses a syntax tree's comparing algorithm to make the matchmaking. In Su et al. (2001), an attribute constraint definition is presented. All the combinations of the possible intervals for all the attributes are calculated to generate the interval records. Matchmaking is based on these interval records to judge whether there is a conflict between two constraint definitions. Sahai (2004) presents a policy constraint based type hierarchical system. The complex environments are treated as high-level resources. By hierarchical system, the policy engine automatically accounts for other dependencies that are created through transitional relationships, and all gathered constraints are checked for validity by an FOL reasoner.

By the efforts of semantic Web research groups, the well-established reasoning tools are a great help to check the validity of the SLO and to build up the matchmaking algorithm for DAML-QoS. The syntax parsers for DAML+OIL (or later OWL) and the DL reasoners can ensure a quick development for the system frameworks. Furthermore, because these tools have already been tested by many research groups, we are more confident about these tools' interpretation for the syntax and semantics of our DAML-QoS specification. When a more optimized reasoner is built, it can be used directly rather than redeveloping all the tools. As to our knowledge, previous works only judge whether two SLOs are compatible or not. In our matchmaking algorithm, SLO compatibility is further divided into five degrees to make a more granular selection.

MODELING

Web services QoS Ontology, especially for service discovery purposes, 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 defining the property and elaborating the property's domain and range constraints; and the metrics layer for metrics definition and measurement. In our prototype, we use DAML+OIL to build up the Ontology. For the purpose of clarity and compactness of this chapter, we will mainly use the DL notions in place of the DAML+OIL syntax for the T-Box definition.

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 follows:

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

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, which makes no sense. Our solution is to use property definition and cardinality to define the QoS constraints. Cardinality is chosen instead of concrete datatype for the following reasons:

- All the QoSProfile layer Ontologies are described in the T-Box to make the description uniform and to 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 that 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 an observer's internal data structures is normally nonnegative integers, such as response time, bit rate, and so forth. Even if the observed data cannot be precisely described by integer domain, through the using of more fine-grained metric unit, this

problem always can be solved. For example, using dollar as the cost unit is not a good solution, but we can change the unit into cents to ensure the correctness. If different QoS metric units for the same property are used in different advertisement, Ontology translation is needed to normalize 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 (Bellwood et al., 2003). It is the service provider's task to set up this layer and provide enough and correct property and cardinality constraint information for a 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 later.

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 that 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:

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

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 is 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. With the range constraints together with the domain constraints, the QoS properties are specified precisely. After the definition of the QoS properties, cardinality constraints are ready to be added on these defined properties in QoS Profile Layer for matchmaking purposes.

The development of QoS Ontology allows a new role of the 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 the 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.

QoS Metrics Layer

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

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

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

The Metric class is a common superclass for all metrics. The Metric class has related properties unit, value, and 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 &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 to initialize the measurement partner's observer. By proper selection of the unit, the nonNegativeInteger is a practical choice for QoS value, as discussed in QoS Profile Layer.

The AtomicMetric's individual definition provides necessary information to initiate the observer. MeasureAt property specifies the observer's location information. Pull and Push are possible ways to report the collected measurement data. Push is the currently utilized report approach. Through the PushPoint property, the AtomicMetric can push the observed QoS data to high-level metrics or the measurement partner. Standard report service interface is defined for the system to work. Since all the data are integer values, this can be achieved more easily.

The complex metrics are composed of other (AtomicMetric or ComplexMetric) metrics. The operand property in ComplexMetric points to these child metrics (AtomicMetrics or ComplexMetric). The function property in ComplexMetric points to the Function class's individual, which describes how to process the operand metrics. The Function class has BooleanFunction, ArithmeticFunction, and AggregateFunction as direct subclasses. The AggregateFunction can process on series of history data. Therefore, the metric aggregation also can be described through ComplexMetric, such as percentile, mean, variance, and frequency. Each ComplexMetric can be viewed as a QoS metric and used as an operand in other ComplexMetric. Through the definition of ComplexMetric, ambiguous understanding of the metric's semantics can be avoided.

Each QoS metric is a subclass of the AtomicMetric or ComplexMetric. The metric's taxonomy is designed by a 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 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.

Basic Profile

Anbazhagan et al. (Mani & Nagarajan, 2002) highlighted that the major requirements for supporting QoS in Web services include Performance, Reliability, Security, and so forth (Weikum, 1999), divided the services QoS into three categories: system centric, process centric, and information centric. The performance, reliability, security, and the like are located in the system centric category. These general QoS metrics are needed normally 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.

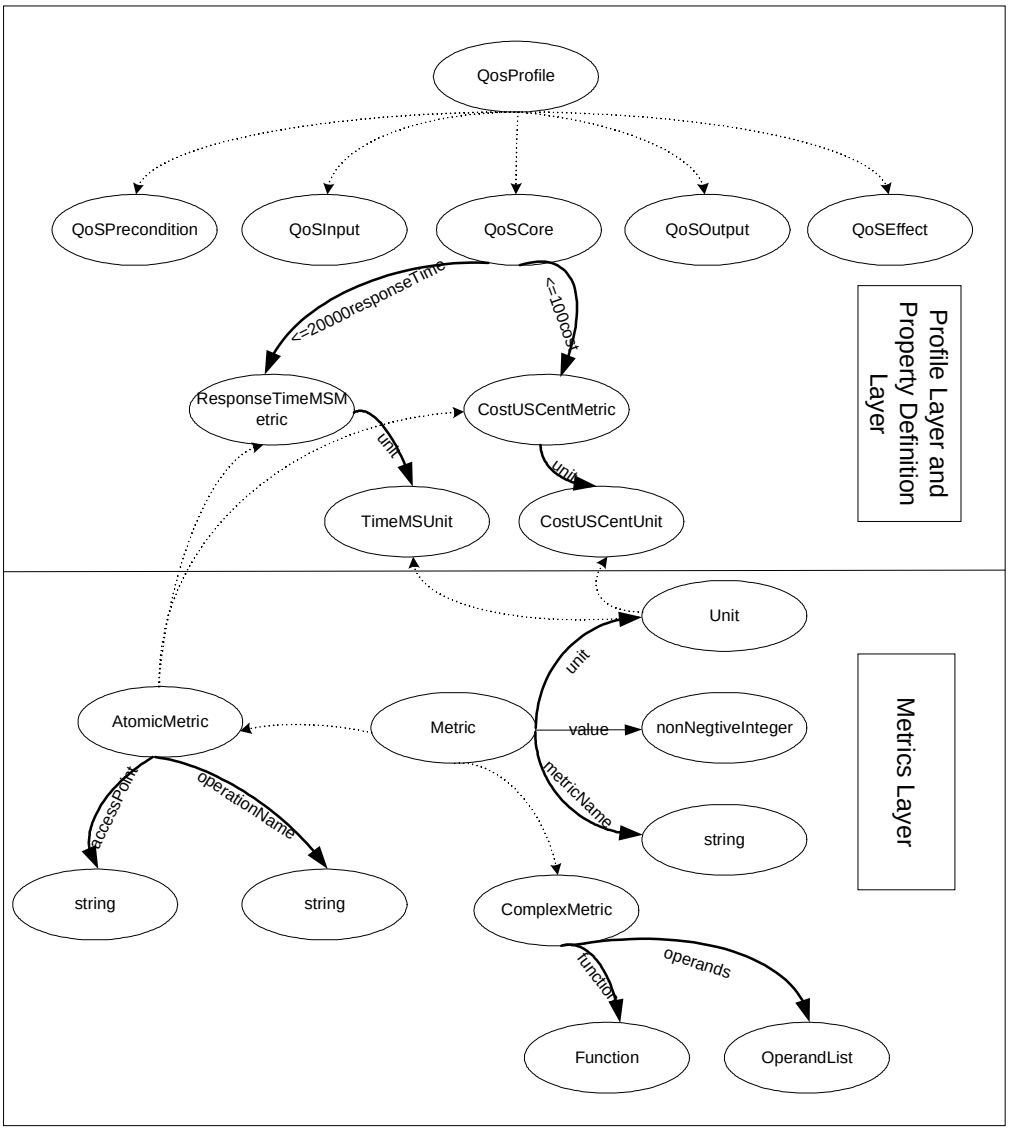
- Response Time is defined as the total time needed by the service requester to invoke the service. It is measured from the time the requester initiates the invocation to the time the requester receives the last byte of the response.
- Cost represents the cost associated with the execution of the service. 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, a service provider and service requester easily can 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 one dollar. In DL syntax, this advertisement can be written as shown in Example 1.

Similar to this 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 five dollars. This inquiry can be expressed as shown in Example 2.

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

Figure 1. Advertisement ontology layers example



In real life, Service Level Agreement can be more complex than the above example. The QoS specification should answer the questions "when, which, where, what, and how the specification should be evaluated" (Sahai et. al., 2002). In our approach, "when and which" questions are answered by the individual definitions for the QoSProfile class. The atomic metric's A-Box definition in the Metrics Layer answers the "where" question. In addition, the Metrics Layer tells the "what and how" question by giving the definitions for Atomic and Complex metrics. Here's an example to show the SLA describing "acceptable response time is <= 40 seconds for 90% of cases, <=35 seconds for 80% of cases, and with variance less than 20%. It is valid from

Example 1.

$$\begin{aligned} Advert \doteq\ & QoSProfile \\ & \sqcap(\leq 100 cost.CostUSCentMetric) \\ & \sqcap(\leq 20000 responseTime.RespMSMetric) \end{aligned}$$

Example 2.

$$\begin{aligned} Inquiry \doteq\ & QoSProfile \\ & \sqcap(\leq 500 cost.CostUSCentMetric) \\ & \sqcap(\leq 40000 responseTime.RespMSMetric) \end{aligned}$$

Example 3.

$$\begin{aligned} AdvertNew \doteq\ & QoSProfile \\ & \sqcap(\leq 40000 responseTimePersentile90.RespMSMetric) \\ & \sqcap(\leq 35000 responseTimePersentile80.RespMSMetric) \\ & \sqcap(\leq 20 responseTimeVariance.Percentage) \end{aligned}$$

Example 4.

$$\begin{aligned} & AdvertNew(AdvertNewInstance) \\ & hasServiceProfile(AdvertNewInstance, \text{"\&provider\_service;\#"}) \\ & startTime(AdvertNewInstance, \text{2004-09-24T09:00:00}) \\ & endTime(AdvertNewInstance, \text{2004-09-30T09:00:00}) \end{aligned}$$

2004-09-24T09:00:00 to 2004-09-30T09:00:00."

The T-Box definition is listed below: the responseTimePersentile90, responseTimePersentile80, and responseTimeVariance should be defined in the Metrics Layer as ComplexMetrics, and they collect the data from responseTime Atomic Metric to generate the proper measurement results (see Example 3).

The Advert class's A-Box contains the separated "when and which" information for the Service Level Objective. This separation helps the concept definition in T-Box to achieve better extensibility. In this A-Box definition example, the first line shows the concept assertion. The next three lines are the role assertions (Example 4).

Percentile, Variance, Mean, FrequencyGreaterThanThreshold, and FrequencyLessThanThreshold often are required in the SLA definition. They are treated as part of the definition of the Complex Metrics.

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 ser-

vices; however, some other services have their own specific QoS properties. This requires that the QoS description for service discovery to have good extensibility to meet different users' 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 achieved naturally in an object-oriented way.

The different scenarios for the creation and maintenance 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, a 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 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. The remaining thing is to set up 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 ensures the reusability of the specification. 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 shown in Example 5.

This Ontology inheritance ability helps to achieve easy extensibility. Meanwhile, the inheritance is a refinement process, which 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 wants to set the response time constraint as ≤ 20000, then this will not take effect. The inherited class still has the response time constraint as ≤ 10000. This will be discussed further in the matchmaking section. Therefore, the template's inheritance should be carefully designed so that the subclasses of the templates are of stricter constraints.

Example 5.

$$\begin{aligned} StorageAdvert1 &\doteq BPTemplate \\ &\quad \sqcap(\geq 10 storage.storageMBMetric) \\ StorageAdvert2 &\doteq BPTemplate \\ &\quad \sqcap(\geq 100 storage.storageMBMetric) \end{aligned}$$

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 the one hand (service profile side), a 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 constraint 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 over the contained QoS property constraints that 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, and so forth. The service requester can choose the most suitable one according to their customized inquiry.

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 subprocesses' QoS metrics. The analytic model used in Cardoso et al. (2002) is a good example.

MATCHMAKING

Matchmaking here is defined as a process that requires a repository to take an inquiry as input and to return all the published advertisements that satisfy the QoS requirements specified in the input inquiry. The service requester uses the same format as the advertisement in their inquiry. The inquiry expresses constraints over aspects of QoS property constraints in which the requester is interested. The inquiry expression will be used to filter out the existing advertisements that are not satisfactory to the requester.

Constraint Order Definition

The novelty of this QoS matchmaking is that it transforms the problem of service QoS constraints comparison into the problem of judging Ontology subsumption relationship. Each service advertisement presents certain combinations of QoS metrics with different quantity constraints. To make a comparison between different advertisements, an order definition among QoS profiles is required. Through the order definition, we can determine whether the guar-

antee given in the provider's advertisement is stricter than the one inquired by the service requester is true.

Cardinality constraints for the property contain ≥ and ≤ operators. These operators on the non-negative integer domain are totally ordered sets. For the same property, the stronger QoS constraint in the description is defined as follows: To ≥ operator, ≥ m means stronger QoS constraint than ≥ n if m > n. To ≤ operator, ≤ m means stronger QoS constraint than ≤ n if m < n. Take response time, for example. ≤10000responseTime:RespMSMetric satisfies the weaker QoS constraint ≤20000responseTime:RespMSMetric because 10000 < 20000. If some metric does not have the non-negative integer values as its domain, it is required to redefine the metric so that the value of the metric locates at the range of the non-negative integer. For example, there are five alphabetical marking levels: A, B, C, D, and F. A possible redefinition of this metric can be 5, 4, 3, 2, 1 for A, B, C, D, F, respectively. According to this definition, all those who pass the examination (i.e., get the marking A, B, C, or D) can be described as ≥ 2.

A stronger constrained QoS profile description always subsumes the weaker one. If this is true, we can transform the problem of service QoS constraints comparison into the problem of judging Ontology subsumption relationship. This can be proved as follows. For example, let's assume profile S and W are stronger and weaker constrained advertisements, respectively. There exist two cases:

Firstly, S and W have 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 ≥ cardinality constraint is used in each case. We have definitions of S and W in Example 6 in which $k_i > n_i$ (S has stronger constrained QoS description), i = 1, 2, …, l. To each property, we have $(\geq k_i P_i.C_i) \sqsubseteq (\geq n_i P_i.C_i)$. By conjunction on these properties, we have $S \sqsubseteq W$.

On the contrary, if $S \sqsubseteq W$ and they have the same property list, S will have the stronger constrained QoS profile description than W. If this is not true (i.e., S has weaker constrained QoS profile description than W), there's some property $(\geq k_t P_t.C_t)$ in S and $(\geq n_t P_t.C_t)$ in W in which $n_t > k_t$ hence $(\geq n_t P_t.C_t) \sqsubseteq (\geq k_t P_t.C_t)$. 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 $x \in \Delta^I$, $x \in (\geq k_t P_t.C_t)^I$, $x \notin (\geq n_t P_t.C_t)^I$ and $x \in (\geq k_i P_i.C_i)^I$ where i = 1, 2, …, l and $i \neq t$. Therefore, $x \in S^I$ while $x \notin W^I$, this contradicts the premise $S \sqsubseteq W$. Therefore, $n_t <= k_t$ and S has the stronger constrained QoS profile description than W.

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

Example 6.

$$S \doteq (\geq k_1 P_1.C_1) \sqcap (\geq k_2 P_2.C_2) \sqcap \ldots \sqcap (\geq k_l P_l.C_l)$$
$$W \doteq (\geq n_1 P_1.C_1) \sqcap (\geq n_2 P_2.C_2) \sqcap \ldots \sqcap (\geq n_l P_l.C_l)$$

Example 7.

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

Matchmaking Algorithm

Formally, the matchmaking can be specified as follows: For a given inquiry *P*, the matchmaking algorithm should return the set of all the published advertisements that are compatible. Two QoS Ontology descriptions (i.e., C1 and C2) are compatible if their intersection is satisfiable (see Example 7).

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 Paolucci et al. (2002) and Li & Horrocks (2003). Some of the definitions are as follows (Li & Horrocks, 2003):

- **Subsume:** If request R is a super-concept of advertisement A (i.e., A $\sqsubseteq$ R), we call the match Subsume; this situation means the request is of weaker constraints than the advertisement.
- **Exact:** If advertisement R and request A are equivalent concepts, this is called the Exact match;
- **Intersection:** If the intersection of advertisement A and request R is satisfiable, we call the match Intersection; that is, $\neg(A \sqcap R \sqsubseteq \bot)$. This means the request and the advertisement still have some similar constraints in common.
- **Disjoint:** Otherwise, it is disjoint (i.e, $A \sqcap B \sqsubseteq \bot$). This means the request and the advertisement totally conflict.

The previous degrees of the match are organized in a corresponding sequence. Subsume matches are considered the preferable match, since we can expect that the advertisement with a more strongly constrained QoS description will subsume the inquiry description; exact matches are the next best, since the advertisement is exactly the same as 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 just means that the advertisement is not incompatible with the inquiry; and Disjoint is the worst case, since it shows that nothing could satisfy both the advertisement and the inquiry, which means a failed match. Intersection matches are not necessarily 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 system to compute a QoSProfile hierarchy for all advertised services. When an inquiry arrives, Racer (Haarslev et al., 2003) 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 subsuming 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 ¬R in the Ontology hierarchy. Advertisements with QoSProfiles subsuming but not equal to ¬R are considered to be Intersection matches, and those subsumed by ¬R are considered to be Disjoint matches. For example, the *Advert* and *Inquiry* defined in Basic Profile section satisfies that Advert ⊑ Inquiry so that the Advert will be included in the result set.

SYSTEM PROTOTYPE AND EXPERIMENT

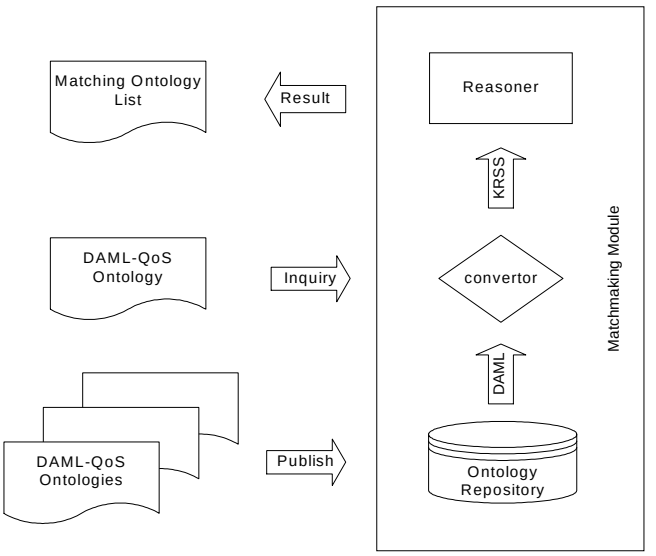
We have made a prototype matching engine that can be used to store, classify, and matchmake the QoS profile Ontologies. In this section, we'll introduce the prototype system and the experiment result for the matchmaking process.

System Prototype

Figure 2 describes the system diagram of our prototype. The prototype system contains an Ontology repository, a converter, and an Ontology reasoner. We have already designed four interfaces for the system: the inquiry interface, the publish interface, the browse interface, and the administration interface. Meanwhile, some tools are chosen to implement our system: Jena parser (HP Labs Semantic Web Programme, 2002) is used to parse the DAML+OIL Ontology; the parsed Ontologies are temporarily stored and processed in the memory through the OilEd's (Bechhofer, 2003) internal Ontology data structure; in the reasoner part, Racer is selected as the Ontology reasoner to classify the Ontologies.

When the service provider publishes its service QoS profile through the publish interface, the Ontology will be parsed by the Jena parser first. If the parsing process ends successfully, the Ontology is stored on the server's Ontology repository. Meanwhile, the Ontology also is loaded into the main memory and represented in the form of the OilEd's internal Ontology data

Figure 2. Matchmaker system prototype



structure. This Ontology data structure is then rendered into Racer's KRSS (Patel-Schneider et al., 1993) description through a special Racer Render. Now we have finished the parsing and the converting process of the Ontology. The Racer engine can accept this KRSS description and keep this in its own knowledge base. By the classification on its knowledge base, the Racer engine reorganizes the Ontologies' hierarchy for inquiry. After these publish processing steps, the service provider receives a unique ID for his published Ontology, and he can manipulate the advertisement through this unique ID in the future.

When the service requester submits an inquiry, a similar parse and conversion procedure happens as the publish procedure. After this, the submitted Ontology is temporarily added into the Racer Ontology knowledge base and classified. As described in the matchmaking section, the matchmaker collects the entire Subsume match list, Exact match list, PlugIn match list, Intersection match list, and Disjoint match list together. Then these five result lists are sent back to the service requester. The submitted Ontology is removed from the Racer knowledge base after the inquiry.

The browse interface enables the service requester to browse the Ontologies list in the repository or view the content of an Ontology by the Ontology name or the unique ID that represents the Ontology. The Administration interface provides the functionality to initialize or reload the whole knowledge base from the repository.

This prototype design has two features: code reuse and extensibility. First, code reuse fastens our development, and each research group focuses on its specialization. Instead of using specially defined constraint syntaxes and a reasoning engine, our Ontology design is based on the DAML+OIL, which has already gained good tool support by different semantic research groups. We easily can reuse most of those tools in our system, such as the Racer reasoner. Once a better reasoning algorithm and its implementation are issued, we easily can plug this new component into our system and enjoy its performance. Second, extensibility is achieved through the parser and render structure. Once a new Ontology definition language such as OWL needs to represent the QoS profile, the framework should not be modified, and only a new parser module for this new Ontology is required to replace the original module. Similarly, a new Reasoner easily can be plugged into the system by introducing a new Render module. The parser and render process is very fast, and the gained extensibility is well worth the slight loss in performance.

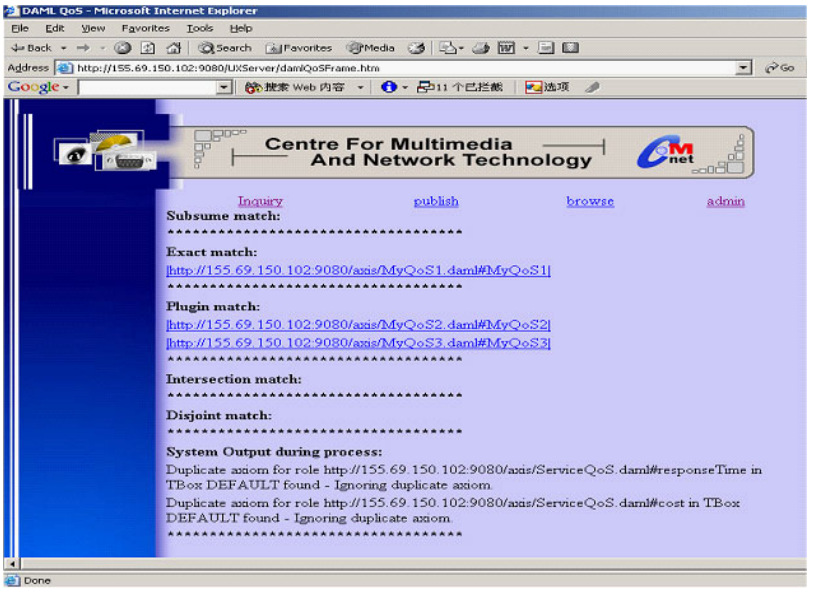
The prototype is wrapped as a Web service, and a corresponding Web page, GUI, is provided for the Web service for ease of use. Following, we'll show an inquiry example to show how the prototype system works.

Suppose the service requester wants to find all the services whose response time is less than 1,001 msecs and the cost is larger than 5,001 U.S. cents. List 1 shows the service requester's inquiry definition. The responseTime and the cost are two object properties whose ranges are TimeMSUnit and CostUSCentUnit, respectively. When this inquiry is issued to the matchmaker, the published services that satisfy the inquiry are added into the corresponding matching degree list and returned back to the requester. Figure 3 shows the returned result of the matched Ontologies. In this example's result, there are one exact match Ontology and two PlugIn match Ontologies. The output of the processing procedure also is shown on the bottom of the Web page.

Listing 1. DAML-QoS inquiry example

```
...
<daml:Class rdf:ID="MyQoS4">
  <rdfs:label>MyQoS1</rdfs:label>
     <rdfs:comment>The inquiry QoS profile definition</rdfs:comment>
     <rdfs:subClassOf>
        <daml:Class rdf:about="&serviceqos;#QoSProfile"/>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
        <daml:Restriction daml:maxCardinality="1001">
           <daml:onProperty rdf:resource="&serviceqos;#responseTime"/>
        </daml:Restriction>
     </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="5001">
      <daml:onProperty rdf:resource="&serviceqos;#cost"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
...
```

Figure 3. Inquiry result GUI



Experiment

The purpose of our experiment is to prove the feasibility of the matchmaking algorithm and to evaluate the performance of our approach. We focus on the matchmaking portion of the system during the experiment. This is the key in the service discovery process when a potentially large number of Web services appear. The testing system for the experiment is Dell PowerEdge Server with Xeon 2.8-GHz

CPU, 1024M memory, and operating system Windows 2003.

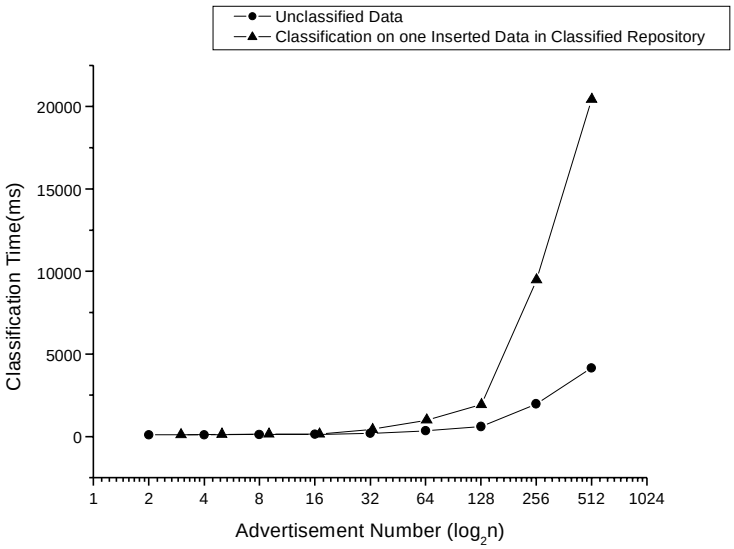
To build up the test environment, an evaluator is written to generate the advertisement Ontologies with given parameters. We have defined five parameters to initiate the evaluator.

1. **Constraint_number** – defines the maximum constraint property number that can appear in the QoS Profile advertisement Ontology. One random number between one and the *constraint_number* is generated for the advertisement Ontology. In the evaluator, this parameter is set at 10. Each constraint property is represented as a property with cardinality, domain, and range constructors on it.
2. **Domain_number** – the number of the property's selectable domains. Five is chosen for this parameter.
3. **Range_number** – the number of the candidate ranges that can be selected for the QoS Property. Each range class is an available metric class defined in the metrics layer. Forty is set for this parameter in the experiment.
4. **Cardinality_number** – helps to generate the cardinality value. The evaluator chooses the cardinality constructor: maxCardinality, minCardinality, or a combination of these two. Then the cardinality value is generated according to the *cardinality_number* parameter. In the evaluator, this parameter is set to 3,000.

After the parameters are set in the evaluator, the evaluator repeatedly generates a new advertisement Ontology and then publishes it to the matchmaker. After publishing a predefined amount of Ontologies, the evaluator sends the classification command to the reasoner, and the classification time is recorded to judge the performance. Figure 4 shows the experiment results of the prototype system's classification time. The x-axis represents the Ontology number in the matchmaker before sending the classification command, while the y-axis represents the classification time for the reasoner. Two series of data are collected. The first data series are the classification time for an unclassified Ontology knowledge base. The second data series is the reclassification time when a new Ontology is inserted into the classified knowledge base.

From the experiment result, we can see that the classification time for the unclassified knowledge base grows smoothly, and the time is acceptable when the Advertisement number is less than or equal to 500. The tableau-based algorithm's transformation rules that handle the at-most restrictions are non-deterministic in the sense that a given A-box is transformed into many finite new A-boxes. Since the at-most rule is frequently applied in our Ontology, the successors of the tree model node are huge. This is a major complexity source of the classification operation. Since the definition in T-Box contains no concept negation and individual definition, it is within DL *ALN*. Structural subsumption algorithm can provide a faster way for classification. The classification time after a new concept is inserted into the classified Ontology knowledge base is longer than a totally new classification. This could be caused by the reclassification time for the breaking down and reorganization of the reasoner's internal data structure. When a more optimized reasoner engine is available, the prototype easily can plug into the new reasoner to improve the system's performance. For example, an optimization algorithm that combines the tableaux with algebraic decision procedure (Haarslev & MÄoller, 2001) is possibly a good solution. From the ex-

Figure 4. Classification time experiment



periment, we can see that the current prototype is suitable for a small or medium-sized discovery repository.

CONCLUSIONS AND FUTURE WORK

This chapter provides a novel DAML-QoS Ontology as a complement for the DAML-S Ontology (Zhou et al., 2004a) to provide a better QoS metrics model. It is designed specially for matchmaking purposes. 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 properties can be added to the basic profiles for certain service categories such as the storage service, computational service, and so forth. One service profile can have multiple QoS profiles representing different service level objectives. A matchmaking algorithm with multiple levels of granularity is presented. A prototype system is designed, based on the Ontology to prove the matchmaking algorithm. An experiment is made based on the prototype to demonstrate its feasibility.

The QoS matchmaking is the 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 (Zhou et al., 2004b). Furthermore, the metrics layer can be studied further to provide better management control together with supporting frameworks.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P.F. (Eds.) (2003). *The description logic handbook: Theory, implementation, and applications*. Cambridge, MA: Cambridge University Press.

Bechhofer, S. (2003). *OilEd*. Retrieved from *http://oiled.man.ac.uk/*

Beckett, D., & McBride, B. (Eds.) (2004). RDF/XML Syntax Specification (Revised). *W3C*. Retrieved from *http://www.w3.org/TR/rdf-syntax-grammar/*

Bellwood, T., Clmen, L., & von Riegen, C. (2003). UDDI Version 3.0.1 Specification.

Cardoso, J., Sheth, A., & Miller, J. (2002). Workflow quality of service. *Proceedings of the International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference*.

Connolly, D., et al. (2001). DAML+OIL (March 2001) Reference Description. *W3C*. Retrieved from *http://www.w3.org/TR/daml+oil-reference*

DAML-S Coalition (2002). DAML-S: Web service description for the semantic Web. *Proceedings of the International Semantic Web Conference (ISWC 02)*.

Dan, A., et al. (2004). Web services on demand: WSLA-driven automated management. *IBM Systems Journal*.

Fox, M., & Gruninger, M. (1994). Ontologies for enterprise integration. *Proceedings of the 2nd International Conference on Cooperative Information Systems (CoopIS)*.

Frolund, S., & Koistinen, J. (1998). *QML: A language for quality of service specification* [Technical Report]. *HPL-98-10*.

Haarslev, V., & MÄoller, R. (2001). Optimizing reasoning in description logics with qualified number restrictions. *Proceedings of the International Workshop on Description Logics (DL-2001)*.

Haarslev V., Moller R., & Wessel M. (2003). RACER: Renamed ABox and concept expression reasoner. *Racer*. Retrieved from *http://www.sts.tu-harburg.de/~r.f.moeller/racer/*

Hondo, M., & Kaler, C. (2002). *Web services policy framework* (WS-Policy) Version 1.0.

Horrocks, I., & Sattler, U. (2001). Ontology reasoning in the SHOQ(D) description logic. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*.

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*.

HP Labs Semantic Web Research (2002). *HP Labs*. Retrieved from *http://www.hpl.hp.com/semweb/*

Jin, J., & Nahrstedt, K. (2004). QoS specification languages for distributed multimedia applications: A survey and taxonomy. *IEEE Multimedia Magazine*, *11*, 74-87.

Li, L., & Horrocks, I. (2003). A software framework for matchmaking based on semantic Web technology. *Proceedings of the International World Wide Web Conference (WWW2003)*.

Ludwig, H., Keller, A., Dan, A., King, R.P., & Franck, R. (2003). Web service level agreements (WSLA) Project. *IBM*. Retrieved from *http://www.research.ibm.com/wsla/documents.html*

Mani, A., & Nagarajan, A. (2002). Understanding quality of service for Web services. *IBM*. Retrieved from, *http://www-106.ibm.com/developerworks/*

library/ws-quality.html?n-ws-1172

Paolucci, M., Kawmura, T., Payne, T., & Sycara, K. (2002). Semantic matching of Web services capabilities. *Proceedings of the First International Semantic Web Conference.*

Patel-Schneider P.F., & Swartout B. (1993). Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. *IBM*. Retrieved from *http://www-db.research.bell-labs.com/user/pfps/papers/krss-spec.ps*

Sahai, A., Durante, A., & Machiraju, V. (2002). Towards automated SLA management for Web services. *HPL-2001-310 (R.1)*.

Sahai, A., Singhal, S.M.V.J.R. (2004). Automated generation of resource configurations through policies. *Policies for Distributed Systems and Networks,* 107-110.

Su, S.Y., et al. (2001). An Internet-based negotiation server for e-commerce. *VLDB, 10*, 72-90.

Tosic, V., Pagurek, B., & Patel, K. (2003). WSOL—A language for the formal specification of classes of service for Web service. *Proceedings of the International Conference on Web Services (ICWS03)*.

W3C Committee (2004). OWL Web ontology language reference. *W3C*. Retrieved from *http://www.w3.org/TR/owl-ref/*

Weikum, G. (1999). Towards guaranteed quality and dependability of information service. *Proceedings of the 8th GI Fachtagung: Datenbanksysteme in Buero, Technik und Wissenschaft*.

Weilai Y., Heiko L.A.D. (2003). Compatibility analysis of WSLA service level objectives. *IBM Research Report.*

Zhou, C., Chia, L.-T., & Lee, B.-S. (2004a). DAML-QoS ontology for Web services. *Proceedings of the International Conference on Web Services (ICWS04)*.

Zhou, C., Chia, L.-T., & Lee, B.-S. (2004b). QoS-aware and federated enhancement for UDDI. *International Journal of Web Services Research (JWSR), 2,* 58-85.

Zinky, J.A., Bakken, D.E., & Schantz, R.E. (1997). Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*.

Chen Zhou received his BE in computer science and technology from Shanghai Jiao Tong University, China (2002). Since then, he has been working toward a PhD in the School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests include Web services discovery, Semantic Web, service QoS, and middleware distributed systems.

Liang-Tien Chia received his BSc and PhD from Loughborough University (1990 and 1994, respectively). He is the director for the Centre of Multimedia and Network Technology and an associate professor in the Division of Computer Communications, School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests are in multimedia storage and retrieval, multimedia processing, error concealment techniques, video communication, bandwidth management, and wireless Internet. He has published more than 50 research papers.

Bu-Sung Lee received his BSc (Honors) and PhD from the Electrial and Electronics Department, Loughborough University of Technology, UK (1982 and 1987, respectively). He is currently an associate professor at the Nanyang Technological University, Singapore. He also holds the position of vice dean of research in the School of Computer Engineering, NTU. He is the technology area director of the Asia Pacific Advance Network (APAN) and an associate with Singapore Research and Education Networks (SingAREN). He has been an active member of several national standards organizations such as the National Infrastructure Initiative (Singapore One) Network Working Group, the Singapore ATM Testbed, and the Bio-Medical Grid (BMG) Task Force. His research interests are in network management, broadband networks, distributed networks, and network optimization.