2008 IEEE International Conference on Web Services

A Semantic QoS-Aware Discovery Framework for Web Services

Qian MA, Hao WANG, Ying LI, Guotong XIE, Feng LIU
IBM China Research Laboratory, Beijing, 100094, China.
E-mail: { maqian, wanghcrl, lying, xieguot, liufcrl }@cn.ibm.com

Abstract

Augmenting web services with explicit semantics forms the foundation of Service Oriented Architectures (SOAs) automation. As more and more Semantic Web Services (SWSs) are deployed, similar SWSs could have quite different quality-of-service (QoS) levels. The QoS-aware discovery becomes an important challenge. While some efforts try to solve it via Constraint Programming (CP), they suffer from the purely syntactic matchmaking method. Furthermore, the construction of constraints and the selection of services are completely dependent on the literal translation from QoS descriptions, which increase obstacles to actually apply CP. In this paper, we propose a semantic QoS-aware framework for SWSs discovery by combining the semantic matchmaking and CP. Initially, a QoS ontology is presented to define QoS data into service descriptions. Then the ontology reasoning is adopted to change previous syntactic matchmaking into a semantic way. Through confirming the compatibility of concepts, complex QoS conditions are solved as constraints and a selection algorithm is proposed to obtain the optimal offer. Finally, the prototype implementation of our framework is discussed and a SWSs discovery case is used to illustrate the comprehensive discovery process.

1. Introduction

Semantic Web Services (SWSs), to enable web services with well-defined semantics, are viewed as a promising technology that provides interoperability between web services by describing their own capabilities in a computer-interpretable way. With the ever increasing number of functional similar SWSs, it is an absolute requirement to distinguish them using a set of quality-of-service (QoS) criteria. In other words, QoS has become an especially important factor in the automatic SWSs discovery [2, 13].

Most approaches on automatic discovery of SWSs use Description Logics (DLs) to semantically match the functional requirements [6, 8], while discovering from the QoS aspect, has not been discussed sufficiently. It not only means to find out services which meet the QoS requirement, but also expects to select the optimal offer by summing up all qualities criteria. Some efforts propose to use Constraint Programming (CP), i.e., to transform as a Constraint Satisfaction Problem (CSP) or a Constraint Satisfaction Optimization Problem (CSOP) which could be solved by checking the conformance of constraints [2, 3]. This is partly due to DL reasoners have limitations on integrating complex QoS conditions within queries. A condition "find a service which *Availability* $\geq$ 0.9, where *Availability* = *MTTF* / (*MTTF* + *MTTR*)"[1] can not be expressed in DLs [3]. However, even if these conditions are able to be described as constraints, there are two major drawbacks in the above efforts. Firstly, they suffer from a purely syntactic way in matching QoS parameters, while the semantic matchmaking is still an indispensable part in the QoS-aware SWSs discovery. The two parties may describe the same QoS concepts in a different way, e.g., "the requester looks for *Price* by dollars, while the provider uses *Cost* by cents". Another more complicated case could be "the requester demands *Availability*, while the provider offers *MTTF* and *MTTR*". Both cases can not be handled by previous approaches. Secondly, if lacking of semantic support, it is actually infeasible to apply CP to solve the QoS-aware discovery. The construction of constraints is subject to the literal translation, but those multifarious data types and features in various parameters, in fact, are hard to be comprehended based on their syntaxes. Moreover, current CP solutions depend on the service requester to provide utility functions for selecting the better QoS offer since the intrinsic data tendencies can not be understood from the syntactic descriptions. This behavior makes an additional burden for using services and an objective evaluation can not be obtained before the intercomparison of available service candidates either. As a conclusion, although QoS conditions are contemplated in previous studies, an integrated discovery framework based on QoS semantics is still an open challenge so far.

To address the challenge, a novel semantic framework for QoS-aware SWSs discovery is proposed in this paper. By introducing semantic technologies and combining with constraint programming, our framework avoids the drawbacks of simply focusing on one aspect and provides an integrated solution. More specifically, it consists of three layers, where semantic technologies construct as the foundation. With an OWL [11]-based QoS ontology that complements OWL-S [12], a promising standard of SWS descriptions, QoS conditions in service requirements and advertisements are ensured to be defined by a common vocabulary. Then, the semantic matchmaking mechanism

[1] MTTF stands for "Mean Time To Failure", while MTTR stands for "Mean Time To Repair".

based on DL reasoning forms the fundamental layer in this framework to examine the compatibility of involved concepts between both sides. After that, QoS conditions are translated as declarative constraints and solved to get eligible offers in the CP layer. Finally, a QoS selection algorithm is adopted in the top layer to obtain the optimal offer by integrating the consideration of all attributes. In order to illustrate the comprehensive discovery process in our framework, we implement a prototype and discuss a SWSs discovery case to compare with existing studies.

The rest of the paper is structured as follows. In Section 2, we discuss related works on discovering SWSs. Section 3 gives an overview of our semantic discovery framework and describes our QoS ontology. The detailed discovery process is studied in Section 4, including the semantic matchmaking, constraint programming and QoS selection. Section 5 presents a prototype implementation and a service discovery case to illustrate our framework. The last section discusses conclusions and future work.

2. Related Work

Semantic discovery of SWSs on the functional aspect is widely studied in many research works. Paolucci et al. [8] define a semantic matchmaking engine to prevent traditional keyword-based search. The "similar degree" is used to rank the matching, i.e., a matching means the offer can be of some use for the requester. In fact, the underlying rationale is performed by the DL reasoning. Li et al. [6] analyze how to use the DAML-S (the OWL-S precursor) and a DL reasoner to implement a service matchmaking prototype. The composite characteristic of QoS parameters is similar with the "correlation" in [14]. Zeng et al. provide a correlation matchmaking algorithm in the services composition. All these works emphasize the functional discovery of SWSs, while we focus on QoS factors. One of our contributions is to integrate semantic technologies into the SWSs discovery on the QoS aspect.

For the QoS-aware discovery, several works target the development of QoS ontology model, such as [9], while not consider QoS matching. Zhou et al. [15] describe QoS conditions by cardinality restrictions within an ontology DAML-QoS and use a DL reasoner to find out proper offers. Unfortunately, complex QoS conditions can not be processed by the DL reasoner. Benbernou et al. [2] include QoS constraints into service descriptions and solve them to get appropriate offers, but the descriptions are described in a private way and constraints are not solved as a formal CSP. Conversely, Martín-Díaz et al.[7] formally describe QoS conditions to constraints and solve them by CP, but in non-SWSs. An improvement is done by Kritikos et al. [5]. Instead of using DL reasoning, they propose a rule-based semantic matchmaking algorithm on a QoS ontology. Meanwhile, a holistic view of the QoS-aware discovery is lacked in that work which leads to the translation between the ontology and constraints rather obscure, especially for handling various data types. Wang et al. [13] present a QoS ontology and selection algorithm to evaluate multiple qualities. However, the matchmaking is missing, so that the selection is possible to base on non-compatible concepts and the symmetric way mentioned in [7] can not be supported without CP.

From the framework viewpoint, the work presented in [3] is a little similar with ours. It proposes a hybrid framework to combine the functional discovery and CP on the QoS aspect for SWSs, which is a subsequent work of [7]. The insufficiency is that it lacks of a semantic foundation, which still induces the QoS matchmaking to a syntactic way. As well as without a semantic vocabulary, it is hard to extract constraints from service descriptions. Another constraint driven framework is METEOR-S [1], which discovers SWSs for dynamic services composition. Our framework could be a complementary work for it to enable QoS considerations during SWSs discovery.

3. Semantic QoS-Aware Discovery Framework

Regarding the drawbacks of purely using CP in the QoS-aware discovery, we present a semantic discovery framework to consider both semantic matchmaking and CP for an integrated solution.

3.1. Framework Overview

The overview of the proposed framework is illustrated in Figure 1. At first, the SWS Requirement that describes a service query is submitted to the discovery framework. In the context of a QoS-aware discovery, this query typically has two parts. One is functional requirements, e.g., the capability descriptions in OWL-S. The other is QoS conditions defined by the QoS ontology. After that, the service query will go through our framework to obtain the optimal advertisement. Finally, each selected SWS Advertisement will be returned to corresponding clients. Its format may need to conform to the specification of a concrete SWS technology for invoking the target service.

Our semantic discovery framework explicitly splits the whole discovery process into three layers. Different from [3], which doesn't prescribe necessary order among each discovery stage in that framework, our discovery process executes in a bottom-to-up way. In more details, the functions of each layer are stated as follows:

1) Semantic Matchmaking Layer: As the semantic technologies are well used in the functional matching of SWSs, this layer applies the DL reasoning into the QoS aspect based on a QoS ontology. Due to DLs' deficiency on processing mathematical operations, the DL reasoning just guarantees that QoS data in the advertisement are semantically compatible with that in the request. For the cases in Section 1, when the requester looks for the expense, the advertisements should have *Price* or *Cost*. If

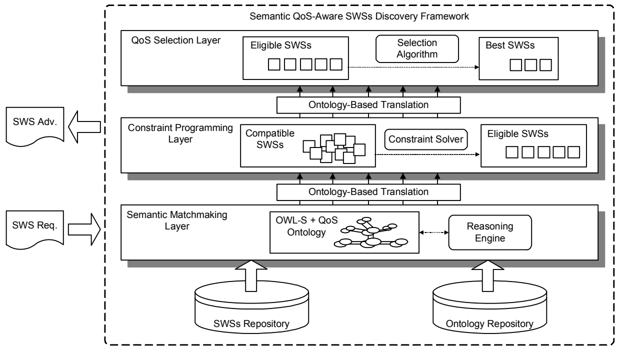

Figure 1. Overview of Semantic QoS-Aware Discovery Framework

it inquires about *Availability*, the offers are expected to provide availability data or values on *MTTF* and *MTTR* which can compute it. This goal of this layer is to prevent the QoS matchmaking from the syntactic way and provide offers owning semantically compatible parameters.

2) Constraint Programming Layer: CP is involved here to check the value conformance of QoS parameters. After identifying compatible concepts between the service requester and provider, our framework translates QoS conditions into a series of constraints according to their intrinsic semantics. In particular, a composite parameter may be calculated by other parameters. Its evaluation is naturally translated as a mathematical expression in CP, which conquers DLs' deficiency on the mathematical calculation. Consequently, QoS constraints are treated as a CSP problem to find out advertisements satisfying the requirements on values.

3) QoS Selection Layer: It is possible that there are many service candidates fulfilling a request from the QoS satisfaction aspect. To obtain the best one can be seen as an optimization task on QoS data. More specifically, our framework quantifies the QoS income of each parameter based on its data tendency from the semantic description, and combines the consideration of all parameters by multiplying their relative weights. The total income is used to sort these advertisements and get the final results.

The QoS-aware discovery process in our framework acts as a filter-by-filter style to select the best offer for a given request. The input of the upper layer comes from the output of the lower layer. All available SWSs and related semantic concepts will be recorded in the SWSs Repository and Ontology Repository.

3.2. QoS Ontology

A QoS ontology is the foundation of our work on the SWSs discovery. Generally, our framework is open to apply any QoS ontologies. Referring to [5, 9], we present a sample QoS ontology here to cover the kernel concepts of QoS parameters. It can be extended to include more concepts. As a start, we formally define the core concept *QoSParameter* in our QoS ontology (See Definition 1).

Definition 1 (*QoS Parameter*). A QoS parameter q in our ontology is defined by the DL notions as follows.

$$\begin{aligned} QoSParameter \equiv\ & (\forall hasCategory.\ Category) \sqcap \\ & (=1\ hasCategory) \sqcap \\ & (\forall hasTendency.\ Tendency) \sqcap \\ & (=1\ hasTendency) \sqcap \\ & (\forall hasMetric.\ Metric) \sqcap \\ & (=1\ hasMetric) \\ Metric \equiv\ & (\forall hasType.\ Type) \sqcap \\ & (=1\ hasType) \sqcap \\ & (\forall hasUnit.\ Unit) \sqcap \\ & (=1\ hasUnit) \sqcap \\ & (\forall hasValue.\ Value) \sqcap \\ & (\leq 1\ hasValue) \end{aligned}$$

where *Category* specifies the measurement aspect of the parameter; *Tendency* is the tendency of the parameter's value; *Metric* provides the metric of the parameter.

The QoS upper ontology based on OWL is shown in Figure 2. Here, we visualize it like a class diagram in the object-oriented approach by EMF Ontology Definition Metamodel (EODM) Workbench in IBM Integrated Ontology Development Toolkit (IODT) [4].

Each *QoSParameter* has a *weight* to present its relative importance. It is restricted to [0, 10]. The sum is 10. Both *Category* and *Tendency* are the union of subclasses. The former has subclasses, e.g., *C_Economic*, *C_Performance*. There are three kinds of *Tendency*. If it is *High*, the client hopes that the value is as large as possible, or else it is *Low*. When the parameter is expected to be close to the value range in the requirement, the tendency is *Given*. If no explicit tendency, we will get *NullTendency*. We have five metric types: *Numeric*, *RegionalNumeric*, *Boolean*, *Enumeration* and *OridnalEnum*. The *RegionalNumeric*'s domain is described in *from* and *to* while *fromInclusive* and *toInclusive* present if they are inclusive. *Enumeration* and *OrdinalEnum* describe a finite collection of items. The values are in *item* and *orderedItem*. The latter one is an ordered collection, e.g. the security may be { *VeryLow*, *Low*, *Medium*, *High*, *VeryHigh* }. The parameter's value is between *start* and *end* in *Value*, while *startInclusive* and *endInclusive* give the boundary. A single value is in the *given* field. Enumerated items will use their literal denotations. Numeric values are parsed when translated into constraints. *Unit* is also the union of subclasses, e.g. *Second*, *Dollar*. *ConversionFormula*s are used to convert into others. We prescribe units for the same measurement to be equal with each other. Take time as an example, we have *Second* ≡ *Minute* ≡ *Hour*. *CompositeQoSParameter* extends *QoSParameter* with a *CompositionFunction*. The function is a mathematical expression and described recursively. It has an *Operator* with two operands, which could be a *NumericOperand*, another *QoSParameter* or another *CompositionFunction*. For complementing OWL-S, a class *QoSProfile* is used to collect all QoS parameters for a service description. The range of *presents* in *Service* is expanded to include *QoSProfile*.

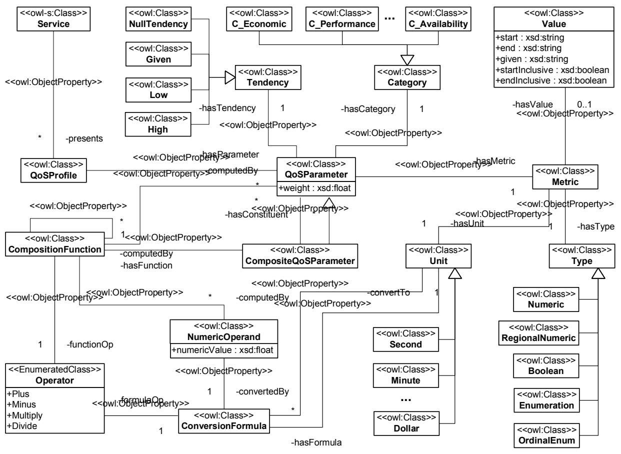

Figure 2. The QoS Upper Ontology

To semantically match QoS parameters (See Section 4.1), we make an assumption to apply our ontology. The constituents for a QoS parameter, such as classes under *Category*, *Tendency*, *Type*, *Value*, and *Unit*, should keep in the same ontology. The requesters and providers can extend parameters by composing them. In particular, it is hard to decide if two *RegionalNumeric*, *Enumeration* or *OrdinalEnum* have the same domains or collected items from DLs. Hence, we use different children to distinguish specific types, e.g. *Fraction* extends *RegionalNumeric* to define [0, 1]. Each domain can be given by the *hasValue* restrictions. All the parameters are finally combined to one ontology for matchmaking. We have provided some extensions for widely-used parameters, e.g. *Availability* is a *CompositeQoSParameter* and calculated by parameters: *MTTF* and *MTTR*. Their definitions are given as follows. The composition function and the conversion formula are ignored due to the limited space. [9, 15] use a similar way to extend their basic QoS ontology model.

$$\begin{aligned}
MTTF \equiv\ & QoSParameter \sqcap \\
& (\forall hasCategory.\ C\_Availability) \sqcap \\
& (\forall hasTendency.\ High) \sqcap \\
& (\forall hasMetric.\ TimeMetric) \\
MTTR \equiv\ & QoSParameter \sqcap \\
& (\forall hasCategory.\ C\_Availability) \sqcap \\
& (\forall hasTendency.\ Low) \sqcap \\
& (\forall hasMetric.\ TimeMetric) \\
Availability \equiv\ & CompositeQoSParameter \sqcap \\
& (\forall hasCategory.\ C\_Availability) \sqcap \\
& (\forall hasTendency.\ High) \sqcap \\
& (\forall hasMetric.\ FractionMetric) \\
Availability \sqsubseteq\ & (\exists\ hasConstituent.\ MTTF) \sqcap \\
& (\exists\ hasConstituent.\ MTTR) \\
TimeMetric \equiv\ & Metric \sqcap \\
& (\forall hasType.\ Numeric) \sqcap \\
& (\forall hasUnit.\ Minute) \\
FractionMetric \equiv\ & Metric \sqcap \\
& (\forall hasType.\ Fraction) \sqcap \\
& (\forall hasUnit.\ NullUnit)
\end{aligned}$$

4. QoS-Aware SWS Discovery Process

Using the QoS ontology as the foundation, we explore the comprehensive process of discovering SWSs in our semantic framework.

4.1. Semantic Matchmaking

A service request usually includes functional and non-functional requirements. Semantic matchmaking on the functional part is widely discussed in many literatures. Here we apply it to non-functional part, typically QoS. A QoS requirement is denoted as $Q_R = \{ q_1, q_2, \ldots, q_m \}$, where $q_1, q_2, \ldots, q_m$ indicates required QoS parameters. Likewise, a QoS advertisement is denoted as $Q_A = \{ q_1, q_2, \ldots, q_n \}$, where $q_1, q_2, \ldots, q_n$ indicates provided QoS parameters. The same as the functional part, the rationale behind the QoS semantic matchmaking is still the DL reasoning. A QoS advertisement matches a request if its parameters can be of some use for the requester. As a consequence, our discovery framework looks for Q_As as many as possible that semantically compatible with Q_R. In formal, the semantic compatibility is defined as:

Definition 2 (*QoS Semantic Compatibility*). A Q_A is semantically compatible with a Q_R, denoted as $Q_A \overset{S}{\cong} Q_R$ iff $\forall q_k \in Q_R, \exists q_j \in Q_A$, q_j is semantically compatible with q_k.

The semantic compatibility between Q_R and Q_A is dependent on the compatibility of their QoS parameters. We argue two differences in our semantic matchmaking for QoS. One is the composition of QoS parameters. Previous compatibility only supports one-to-one matching between concepts. Nevertheless, a composite parameter may be computed by other parameters, and hence the complicated example about *Availability*, *MTTF* and *MTTR* in Section 1 can not be handled in these methods. To overcome this barrier, our algorithm enables one-to-multiple matching by using the semantics recorded by the composite QoS parameter in our ontology. The other is the "subsumption" between two concepts. Different from previous methods using of this relationship to rank the matching degree, we propose that the matching degree of QoS is determined by the conformance on parameters' values (See Section 4.2). We do not restrict either side to extend the ontology, while our algorithm guarantees the compared concepts from two parties having potential semantic relationships. Thus, the semantic compatibility between QoS parameters is enlarged as:

Definition 3 (*Enlarged QoS Parameter Compatibility*). There is a QoS parameter $q_j \in Q_A$ semantically compatible with $q_k \in Q_R$, denoted as $q_j \overset{S}{\cong} q_k$, iff in the ontology, either 1) $q_j \equiv q_k$; 2) $q_j \sqsubseteq q_k$ or $q_k \sqsubseteq q_j$; 3) q_k can be composed by $Q = \{ q_1, q_2, \ldots, q_s \}$ via a composition function f, where $\forall q_y \in Q, \exists q_x \in Q_A$ ($q_x \overset{S}{\cong} q_y$) based on 1) 2). If none of q_x satisfies the compatibility and q_y is a composite parameter, the decomposition continues recursively.

Our QoS semantic matchmaking algorithm is listed as follows (See Algorithm 1). It accepts the requirement and

Algorithm 1: $\boldsymbol{Q'_A}$ = semanticQoSMatchmaking (Q_R, $\boldsymbol{Q_A}$), where $\boldsymbol{Q_A} = \{ Q_{A1}, Q_{A2}, \ldots, Q_{Au} \}$, $\boldsymbol{Q'_A} \in \mathrm{P}(\boldsymbol{Q_A})$

```
Boolean match = FALSE;
for each $Q_{Ai} \in \boldsymbol{Q_A}$ do
  $Q'_R \leftarrow Q_R$;
  for each $q_k \in Q'_R$ do
    match = FALSE;
    for each $q_j \in Q_{Ai}$ do
      if isSemanticCompatible ($q_j$, $q_k$) then
        match = TRUE; break;
      end if
    end for
    if match == FALSE then
      if $q_k \sqsubseteq$ CompositeQoSParameter then
        $Q'_R$.append (getConstituents ($q_k$));
      else break;
      end if
    end if
  end for
  if match == TRUE then $\boldsymbol{Q'_A}$.append($Q_{Ai}$); end if
end for
return $\boldsymbol{Q'_A}$;
```

a set of advertisements as inputs. In the loop, each QoS parameter in the offer will compare with a parameter of the demand to decide their compatibility. If a parameter can not be matched and it is a composite parameter, the algorithm splits it and appends all its constituents into the queue waiting for comparison. Finally, the algorithm outputs a subset of offers containing compatible ones.

4.2. Constraint Programming

Although [15] proposes to use cardinality restrictions to depict QoS requirements, DL reasoners fail to process mathematical operations. Thus, CP is used here to check the QoS conformance as a CSP (See Definition 4). QoS conditions are translated to declarative constraints.

Definition 4 (*Constraint Satisfaction Problem*) A CSP is defined as the tuple $p = < V, D, C >$, where

- $V = \{ v_1, v_2, \ldots, v_n \}$ is a set of variables;
- $D = \{ d_1, d_2, \ldots, d_n \}$ is a set of nonempty domains corresponding to each v_k in V;
- $C = \{ c_1, c_2, \ldots, c_m \}$ is a set of declarative constraints. Each $c_k = < V_{ck}, D_{ck}, R_{ck} >$, where $V_{ck} = \{ v_i, v_{i+1}, \ldots, v_j \} \subseteq V$, $D_{ck} = \{ d_i, d_{i+1}, \ldots, d_j \} \subseteq D$ and one restriction $R_{ck} \subseteq d_i \times d_{i+1} \times \ldots \times d_j$.

One of the difficulties to apply CP is the construction of constraints from service descriptions, which is merely a literal way in [3]. In our framework, the QoS ontology can guide this construction. A *QoSParameter* is mapped to v in the constraints, while its metric type represents the domain d. The value restriction could be concluded as:

- *Numeric* and *RegionalNumeric* Types: The *start* and *end* with *startInclusive* and *endInclusive* in the *Value* indicate the specific range of the numeric metric or the property *given* specifies a certain value.
- *Boolean* Type: TRUE/FALSE is mapped to 1/0.
- *Enumeration* Type: Use the requirement to align the collected items in the advertisement. Map each item to 1/0 according to whether it appears. The comparison of parameter v becomes to determine whether there is a set of corresponding items having the same value.
- *OrdinalEnum* Type: Map the literal domain to a numeric domain. Then construct the restriction based on corresponding values, e.g. the security is { *VeryLow*, *Low*, *Medium*, *High*, *VeryHigh* }, so it is mapped to { *0*, *1*, *2*, *3*, *4* }. The security v better than *Medium* is $v \geq 2$.

Usually, if we have q_k in the requirement and q_j in the advertisement such that $q_j \overset{S}{=} q_k$, we will have a constraint $v_j = v_k$, where v_j and v_k are the corresponding variables to q_j and q_k. There are two places involving mathematical operations: the composite QoS parameter and the unit conversion. We will rewrite them to separate constraints, e.g. "find a service which *Availability* $\geq$ 0.9, where *Availability* = *MTTF* / (*MTTF* + *MTTR*)" is going to be:

- $c_1 = < V_{c1}, D_{c1}, R_{c1} > = < \{ Availability \}, \{ [0, 1] \}, \{ Availability \geq 0.9 \} >$
- $c_2 = < V_{c2}, D_{c2}, R_{c2} > = < \{ Availability, MTTF, MTTR \}, \{ [0, 1], [0, +\infty], [0, +\infty] \}, \{ Availability = MTTF / (MTTF + MTTR) \} >$

Another case "the requester looks for *Price* by dollars, while the provider uses *Cost* by cents" is turn to be:

- $c_3 = < V_{c3}, D_{c3}, R_{c3} > = < \{ Price, Cost \}, \{ [0, +\infty], [0, +\infty] \}, \{ Cost = Price * 100 \} >$

The QoS conformance lies in determining whether every solution to the advertisement's CSP can also be a solution to the requirement's CSP (See Definition 5 [7]).

Definition 5 (*CSP Conformance*) In the CP layer, an advertisement conforms to a requirement iff

Conformance (C_A, C_R) $\leftrightarrow$ Satisfy (C_A, $\neg C_R$) = $\varnothing$

where C_A and C_R are the constraints of the advertisement and requirement respectively. The Satisfy(x, y) is to check if a successful assignment to x is also a solution to y.

The essence of the conformance is to decide whether the guarantee in the advertisement is no looser than that in the requirement. The CP algorithm (See Algorithm 2) is a bridge from the semantic matchmaking to QoS selection (See Section 4.3). We use the degree of "no looser" to rank the output of our algorithm, whose definition is different from [8, 15]. The highest is Exact that means the constraints in the advertisement are no looser than those in the requirement, either with better QoS or equal. Plugin match is the second preferable since the advertisement provides looser constraints, so the values in the looser region could not satisfy the requirement. Intersection match is the next best one since we can just expect that the intersection part of the solution space between the requirement and advertisement could be used. Disjoint is the lowest level since it shows that nothing satisfies both parties. In other words, it is a failed match.

4.3. QoS Selection

Algorithm 2: qosCSPSolving (Q_R, $\boldsymbol{Q'_A}$), where $\boldsymbol{Q'_A}$ = { Q'_{A1}, Q'_{A2}, …, Q'_{Av} }

```
1:   $C_R \leftarrow$ translateToConstraints ($Q_R$);
2:   for each $Q'_{Ai} \in \boldsymbol{Q'_A}$ do
3:       $C_A \leftarrow$ translateToConstraints ($Q'_{Ai}$);
4:       if Satisfy ($C_A$, $\neg C_R$) = $\varnothing$ then
5:           $Q'_{Ai}$.rank = Exact;
6:       else if Satisfy ($\neg C_A$, $C_R$) = $\varnothing$ then
7:           $Q'_{Ai}$.rank = Plugin;
8:       else if $\neg$ Satisfy ($C_A$, $C_R$) = $\varnothing$ then
9:           $Q'_{Ai}$.rank = Intersection;
10:      else $Q'_{Ai}$.rank = Disjoint;
11:      end if
12:  end for
```

It is possible that several advertisements are ranked Exact after our CP checking. Our ultimate goal is to get the best choice from the client's viewpoint. It is a CSOP for evaluating all quality metrics in combination. The selection is based on each parameter's utility function u: $d \rightarrow [0, 1]$, where d is its domain. Previous studies, such as [3, 5], depend on the requester to provide such functions, which makes an additional burden for using services. Moreover, it is also infeasible to do a fair evaluation before the comparison of various candidates since the maximal value span of each parameter is not known. That span, however, is used to normalize the QoS income [13]. Thus, we present a more reasonable way to select the final result based on data tendencies in our QoS ontology.

● Preprocessing Step: We assume that $\boldsymbol{Q''_A}$ = { Q''_{A1}, Q''_{A2}, …, Q''_{At} } is the advertisements with Exact rank and each Q''_{Ai} = { q_{i1}, q_{i2}, …, q_{in} }. To construct this, we select a representative value for each parameter if its value is a range. We use the worst case. For *Numeric* and *RegionalNumeric*, considering $a \le q_{ij} \le b$, a is selected if *Tendency* is *High*. Otherwise, b is the choice if *Tendency* is *Low*. When *Tendency* is *Given*, it means that the data should be close to the middle of the given region in the requirement, so we will use (a + b) / 2. *Enumeration* and *Boolean* do not have "better" semantics, so they are not taken into account. Finally, *OrdinalEnum* is transformed into the numeric value and handled as the same way.

Due to the composite QoS parameters and the units' conversions, we need to align Q''_{Ai} with Q_R by using composition functions and conversion formulas. Given Q_R = { q_1, q_2, …, q_m }, we get a matrix M_A.

$$M_A = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ q_{t1} & q_{t2} & \cdots & q_{tm} \end{bmatrix}$$

● Normalization Step: Each QoS parameter would have its value span among advertisements, and hence it is hard to make a fair evaluation. This step normalizes them in [0, 1] to guarantee they are evaluated by the same span.

1) If the *Tendency* is *High*, the ratio is calculated by the following formula.

$$q'_{ij} = (q_{ij} - q_{\min}) / (q_{\max} - q_{\min})$$

2) If the *Tendency* is *Low*, the ratio is calculated by the following formula.

$$q'_{ij} = (q_{\max} - q_{ij}) / (q_{\max} - q_{\min})$$

3) If the *Tendency* is *Given*, the ratio is calculated by the following formula.

$$q'_{ij} = (\alpha - \left| q_{ij} - (g_{start} + g_{end})/2 \right|) / (\alpha - \beta)$$

where q_{max} = max { q_{ij} }, q_{min} = min { q_{ij} }, g_{start}, g_{end} are the *start*, *end* of given range in the requirement, α = max { $\left| q_{ij} - (g_{start} + g_{end})/2 \right|$ }, β = min { $\left| q_{ij} - (g_{start} + g_{end})/2 \right|$ } for $\forall j \in$ { $1, 2, ..., m$ }.

● Combination Step: Different users have different preferences. It can be projected as the relative importance of each QoS parameter. For instance, the service selection may be driven by price, called price-sensitive, regardless of other qualities. Conversely, a requester may be service-sensitive, who considers more on other qualities than the price. The *weight* describes such relative importance. We assume W = { w_1, w_2, …, w_m } as the vector of weights. The combination of all parameters is given as follows.

$$M''_A = M'_A \times W = \sum_{j=1}^{m} (q'_{ij} \times w_j)$$

5. Implementation and Case Study

5.1. Prototype Overview

To demonstrate our semantic discovery framework, we have implemented a prototype of QoS-aware discovery engine based on it, as shown in Figure 3. The ontology development environment uses the IODT, which includes EODM to manipulate an ontology using Java objects and an OWL repository to save our QoS ontology. It has a reasoner to support taxonomy subsumption reasoning for the ontology. When *Query Receiver* receives a request, it would extract the QoS requirement, and then look for the advertisements with compatible QoS concepts during the semantic matchmaking. After that, *Constraint Translator* would translate the QoS conditions in compatible offers into a series of declarative constraints. They are described by the Oz language and solved by Mozart programming system [10]. Towards QoS advertisements that are ranked as Exact match, *QoS Selector* will choose the best offer according to our three-step QoS selection algorithm. At last, the result is returned to the requester. The prototype has a SWSs repository to contain all the advertisements published by the service provider.

5.2. Case Study

For comparison, we have conducted an experiment in our prototype based on the case in [7]. They implement a

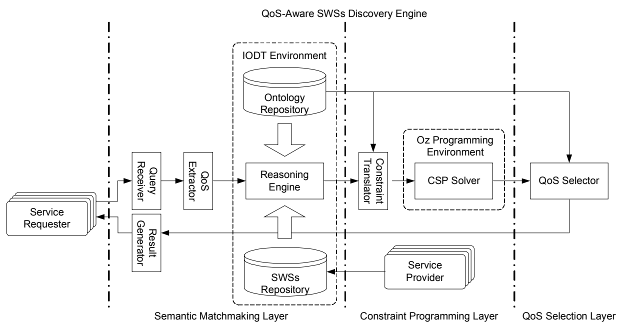

Figure 3. Prototype Architecture

hypothetical web portal to look for web services that deliver video on demand. Different video providers offer discrepant QoS conditions. The web portal not only looks for services that fulfill QoS requirements, but also selects the best one for a request. [7] considers two parameters. One is *Availability* that is computed as *MTTF* / (*MTTF* + *MTTR*). The other is a domain specific parameter *Media Support* which means the supported connection mode, i.e. { *Modem*, *ISDN*, *ADSL* }. To show our capabilities, we increase: *Price/Cost*, *Security*, *Response Time*, *Buffer Time*, *Reputation*, and *Exception Handling*, whose types are *Numeric*, *OrdinalEnum* { *VeryLow*, *Low*, *Medium*, *High*, *VeryHigh* }, *Numeric*, *Numeric, RegionalNumeric* [0, 5] and *Boolean*. Each is possible to be a single value or a range. For demonstrating the semantic matchmaking, besides the availability case, we assume that the requester demands *Price* by *Dollar*, while some offers provide *Cost* by *Cent*. The complete service data are in Table 1.

We start from the semantic matchmaking. The QoS requirement and advertisement is defined as follows. We merely show *Price* and *Cost* here for matchmaking, while ignore others due to the limited space. The descriptions of the *Availability*, *MTTF* and *MTTR* are in Section 3.2.

$$
\begin{aligned}
QoSProfile_{req} \sqsubseteq\ & QoSProfile \sqcap \\
& (\exists hasParameter.\ Availability) \sqcap \\
& (\exists hasParameter.\ Price) \sqcap\ \ldots \\
QoSProfile_{adv} \sqsubseteq\ & QoSProfile \sqcap \\
& (\exists hasParameter.\ MTTF) \sqcap \\
& (\exists hasParameter.\ MTTR) \sqcap \\
& (\exists hasParameter.\ Cost) \sqcap\ \ldots \\
Price \equiv\ & QoSParameter \sqcap \\
& (\forall hasCategory.\ C\_Economic) \sqcap \\
& (\forall hasTendency.\ Low) \sqcap \\
& (\forall hasMetric.\ PriceMetric) \\
Cost \equiv\ & QoSParameter \sqcap \\
& (\forall hasCategory.\ C\_Economic) \sqcap \\
& (\forall hasTendency.\ Low) \sqcap \\
& (\forall hasMetric.\ CostMetric) \\
PriceMetric \equiv\ & Metric \sqcap \\
& (\forall hasType.\ Numeric) \sqcap \\
& (\forall hasUnit.\ Dollar) \\
CostMetric \equiv\ & Metric \sqcap \\
& (\forall hasType.\ Numeric) \sqcap \\
& (\forall hasUnit.\ Cent)
\end{aligned}
$$

In this way, the framework does not care the syntactic name of these parameters, but focuses on the semantic compatibility. *Availability* is compatible with *MTTF* and *MTTR* since it is composed by them.

We then translate the QoS conditions to a series of constraints. The sample Oz code is provided as follows to check the conformance. A_5 is ranked Fail since its *Price* and *Exception Handling* do not meet the requirement. Due to the ranges of *Media Support* and *Response Time* are intersected with the requirement, A_6 is decided as Intersection. A_7 has got a Plugin score for *Availability* and *Security* looser than the demand. Others are Exact match.

To select the best one, we consider the value tendency. *Availability*, *MTTF*, *Security* and *Reputation* are expected to be higher, while *MTTR*, Price/*Cost* and *Response Time* are to be lower. *Buffer Time* is required to be close to the given range since it will influence the video's fluency if it is too long or too short. *Availability* could be computed by *MTTF* and *MTTR*. We get the matrix M_A.

$$
M_A = \begin{bmatrix} 0.92 & 1.50 & 3 & 4.0 & 80 & 2.8 \\ 0.94 & 2.30 & 3 & 3.6 & 45 & 3.3 \\ 0.96 & 2.50 & 4 & 1.2 & 60 & 5.0 \\ 0.99 & 3.00 & 2 & 2.3 & 90 & 2.0 \end{bmatrix}
$$

After the normalization step, we get the matrix M'_A.

$$
M'_A = \begin{bmatrix} 0 & 1 & 0.500 & 0 & 0.333 & 0.267 \\ 0.286 & 0.467 & 0.500 & 0.143 & 0.500 & 0.433 \\ 0.571 & 0.333 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0.607 & 0 & 0 \end{bmatrix}
$$

The weights vector *W* defines the relative importance. We do a comparison between the price-sensitive and service-sensitive selection. For price-sensitive, *W* is { 1, 7, 0.5, 0.5, 0.5, 0.5 }. The proportion between the price and other qualities is 7:3. Conversely for service-sensitive, *W*

Table 1. Experiment Data

Data	Availability		Media Support	Price / Cost	Security	Response Time (s)	Buffer Time (s)	Reputation	Exception Handling
	MTTF (m)	MTTR (m)							
R	≥ 0.90		Modem, ISDN	≤ $3.00	≥ Medium	≤ 4.0	[30, 90]	≥ 2.0	True
A_1	[110, 120]	[5, 10]	Modem, ISDN, ASDL	150C	≥ High	[2.5, 4.0]	[70, 90]	2.8	True
A_2	[180, 200]	[5, 12]	Modem, ISDN	$2.30	≥ High	[1.5, 3.6]	[30, 60]	3.3	True
A_3	≥ 0.96		Modem, ISDN	250C	≥ Very High	[0.8, 1.2]	60	5.0	True
A_4	≥ 0.99		Modem, ISDN, ASDL	$3.00	≥ Medium	≤ 2.3	90	2.0	True
A_5	[240, 280]	[8, 20]	Modem, ISDN	$4.00	≥ Medium	≤ 3.0	[30, 60]	3.0	False
A_6	[150, 180]	[10, 15]	ISDN, ADSL	180C	≥ Very High	[2.5, 5.0]	[40, 50]	3.0	True
A_7	≥ 0.86		Modem, ISDN	200C	≥ Low	≤ 3.0	75	2.8	True

```
proc {QoS Root} Availability MTTF MTTR Price Cost ... in
  Root = sol (avail:Availability mttf:MTTF mttr:MTTR price:Price
  cost:Cost …)
  %Transform all variables into positive integers.
   Availability :: 0#100     %Similar definitions on MTTF, MTTR
   …                         %Cost, Price and other parameters.
  MTTF >=: 110               %QoS in the advertisement A1.
  MTTF =<: 120
  MTTR >=: 5
  MTTR =<: 10
  Cost =: 15000
  Availability <: 90         %QoS in the opposition of the requirement
   Price >: 300
  %Oz has no finite domain propagators for fractions, thus multiply
  %with the denominators.
  Availability * (MTTF + MTTR) =: MTTF * 100
  Cost =: Price * 100
  ...                          %Omit other parameters.
  { FD.distribute ff Root }
end
```

is { 2, 3, 1, 1, 1, 2 }, where the price and other factors is in the ratio of 3:7. The evaluation is shown in Table 2.

According to the evaluation, A_1 is the best one for price-sensitive selection. Although most parameters in A_2 are better than A_1, it can not win since its price is higher. But *Price* can not dominate the evaluation. A_3 precedes A_2 since other parameters are more valuable. On the contrary, A_3 has the highest score in the service-sensitive evaluation, but other qualities can not dominate the selection either. A_1 is better than A_2 since it has a more attractive price.

Table 2. Results of Sensitivity Adjustment

Adv	Price-Sensitive QoS Value	Service-Sensitive QoS Value
A_1	7.550	4.367
A_2	4.343	3.982
A_3	4.902	7.141
A_4	1.304	2.607

6. Conclusion and Future Work

QoS-aware discovery can be deemed as one of the important challenges for SWSs. To our best knowledge, it still lacks of a comprehensive discovery framework. This paper proposes a framework by combining the semantic matchmaking and CP. A QoS ontology is used as the semantic foundation to prevent the syntax-based search. After the matchmaking, QoS conditions are translated to constraints and solved as a CSP. A selection algorithm is used to obtain the best offer for a QoS requirement. At last, we present a prototype and adopt a discovery case to evaluate our framework. Our future work will integrate the framework into existing implementations of service registry, e.g. UDDI, and test its performance. We also want to investigate more in future on how to deal with the services whose QoS parameters are not exactly matched.

References

[1] R. Aggarwal, K. Verma, J. Miller, W. Milnor. Constraint Driven Web Service Composition in METEOR-S. In 1st International Conference on Services Computing (SCC'04), 2004.

[2] S. Benbernou, M.-S. Hacid. Resolution and Constraint Propagation for Semantic Web Services Discovery. In Distributed and Parallel Databases, vol. 18, no. 1, pp. 65-81, 2005.

[3] J.M. García, D. Ruiz, A. Ruiz-Cortés, O. Martín-Díaz, M. Resinas. A Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming. In 5th International Conference on Service-Oriented Computing (ICSOC'07), 2007.

[4] IBM AlphaWorks. IBM Integrated Ontology Development Toolkit, 2006. http://www.alphaworks.ibm.com/tech/semanticstk

[5] K. Kritikos, D. Plexousakis. Semantic QoS Metric Matching, In 4th IEEE European Conference on Web Services (ECOWS'06), 2006.

[6] L. Li, I. Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology, In 12th International World Wide Web Conference (WWW'03), 2003.

[7] O. Martín-Díaz, A. Ruiz-Cortés, D. Benavides, A. Durán, M. Toro. A Quality-Aware Approach to Web Services Procurement, In 4th International VLDB Workshop Technologies for E-Services (TES'03), 2003.

[8] M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara. Semantic Matching of Web Services Capabilities. In 1st International Semantic Web Conference (ISWC'02), 2002.

[9] I.V. Papaioannou, D.T. Tsesmetzis, I.G. Roussaki, M.E. Anagnostou. A QoS Ontology Language for Web-Services, In 20th International Conference on Advanced Information Networking and Applications (AINA'06), 2006.

[10] C. Schulte, G. Smolka. Finite Domain Constraint Programming in Oz: A Tutorial, 2006. http://www.mozart-oz.org/documentation/fdt/

[11] W3C. OWL: Web Ontology Language, 2004. http://www.w3.org/TR/owl-ref/,

[12] W3C. OWL-S: Semantic Markup for Web Services, 2004. http://www.w3.org/Submission/OWL-S/

[13] X. Wang, T. Vitar, M. Kerrigan, I. Toma. A QoS-aware Selection Model for Semantic Web Services. In 4th International Conference on Service-Oriented Computing (ICSOC'06), 2006.

[14] L. Zeng, B. Benatallah, G.T. Xie, H. Lei. Semantic Service Mediation. In 4th International Conference on Service-Oriented Computing (ICSOC'06), 2006.

[15] C. Zhou, L.-T. Chia, B.-S. Lee. DAML-QoS Ontology for Web Services. In 2nd International Conference on Web Services (ICWS'04), 2004.