A Semantic QoS-Based Web Service Discovery Engine for Over-Constrained QoS Demands

Kyriakos Kritikos and Dimitris Plexousakis

Institute of Computer Science, FORTH, Greece
{kritikos,dp}@ics.forth.gr

Abstract. The success of the Web Service (WS) paradigm has led to a proliferation of available WSs. Semantic discovery mechanisms have been invented to overcome UDDI's syntactic discovery solution by providing more precise results. However, the problem remains as many functionally-equivalent WSs are returned. Its solution comes in terms of semantic QoS-based description and discovery of WSs. We have already presented a rich and extensible ontology language for QoS-based WS description that is called OWL-Q and we have proposed a semantic QoS metric matching algorithm. Based on this algorithm, we have extended a Constraint-Programming-based approach for QoS-based WS discovery. In this paper, we show an extension of OWL-Q with SWRL rules and propose a modification to the metric matching algorithm to make it more feasible. Moreover, we propose and analyze an automated approach for semantic QoS-based WS discovery that provides solutions even for over-constrained QoS-based WS demands.

1 Introduction

The success of the Web Service (WS) paradigm has led to a proliferation of available WSs. Current WS standard technologies involve the advertisement of static functional descriptions of WSs in UDDI registries, leading to a WS discovery process that returns many irrelevant or incomplete results. While semantic functional discovery approaches (e.g., [1]) have been invented to overcome this problem, the amount of functionally equivalent WS advertisements returned is still large. The solution to this problem is: a) the description of the Quality of Service (QoS) aspect of WSs, which is directly related to their performance; b) filtering of WS functional discovery results based on user constraints on their QoS descriptions; c) sorting the results based on user weights on QoS metrics.

QoS of a WS is a set of non-functional attributes that may impact the quality of the service offered by the WS. Each QoS attribute is measured by one or more QoS metrics, which specify the measurement method, schedule, unit, value range and other measurement details. A QoS specification of a WS is materialized as a set of constraints on a certain set of QoS metrics. These constraints restrict the metrics to have values in a certain range or in a certain enumeration of values. Actually, the current modeling efforts of QoS specifications only differ in the expressiveness of these constraints. However, these efforts fail in QoS

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 151–164, 2009.

metrics modeling. The main reason is that their QoS metric model is syntactic, poor and not extensible. In this way, the most prominent QoS-based WS discovery algorithms [2,11], produce irrelevant or incomplete results. Moreover, these prominent algorithms have their own shortcomings that will be analyzed later in this paper.

Based on the above deficiencies, we have developed OWL-Q [3], a rich, extensible and modular ontology language that complements the WS functional description language OWL-S. In addition, we have developed a QoS metric matching algorithm that infers the equivalency of two different QoS metric descriptions based on the mathematical equivalency of their derivation formulas. Based on OWL-Q and the metric matching algorithm, we have extended the discovery algorithm of [2]. In this paper, we firstly review the state-of-the-art in QoS-based WS description and discovery. Then, we explain that OWL cannot be used for reasoning about relations between properties so as to justify the extension of OWL-Q with SWRL (`http://www.w3.org/Submission/SWRL`) rules. We also describe some modifications that have been made to OWL-Q. Next, we explain Constraint Programming (CP) [4] as it is the main technique used for matchmaking QoS metrics and QoS specifications. Then, we clarify the way the QoS metric matching algorithm has been modified to compute the satisfiability of possibly non-linear Constraint Satisfaction Problems (CSPs) [4] instead of inferring mathematical equivalency of formulas. In this way, this algorithm becomes computationally feasible. Next, we propose a complete and automated CP-based QoS-based WS Discovery algorithm that provides solutions even in cases where the QoS-based WS demand is over-constrained. Then, we analyze the architecture and functionality of our QoS-based WS Discovery Engine, which is currently under development. Finally, we conclude by drawing directions for further research.

2 Related Work

The *WSDL* and *UDDI* WS standards are *syntactical* approaches that do not express the QoS aspect/part of WS Description. Although *OWL-S* is a standard *semantic* approach for WS Description, its support for QoS description is limited and not well-defined.

Ran [5] proposes a syntactic extension to UDDI for QoS-based WS description. Maximilien and Singh [6] present an architecture and a conceptual model of WS reputation that does not include concepts like QoS constraints, offers and demands. Furthermore, the QoS metrics model is not rich enough. Tosic, Pagurek et. al. [7] present the XML-based *Web Service Offerings Language* (WSOL). Their work comes with the following shortcomings: (a) no specification of a QoS demand; (b) metrics ontologies are not developed. *Web Service Level Agreement* (WSLA) [8] is a XML language used for the specification of Service Level Agreements (SLAs). It represents a purely syntactic approach that is not accompanied by a complete framework. Tian et. al. [9] propose an ontology-based approach for QoS-based WS description. However, not only there is no complete and accurate

description of QoS constraints, but also metrics ontologies are only referenced. Oldham et. al. [10] offer a semantic framework for the definition and matching of *WS-Agreements*. However, only unary QoS metric constraints can be expressed while metric matching can only be enforced by manual incorporation of rules.

Zhou et. al. [11] extend OWL-S by including a QoS specification ontology. In addition, they propose a novel matchmaking algorithm, which is based on the concept of *QoS profile compatibility*. The deficiencies of this research effort are the following: (a) The metrics model is not rich enough; (b) QoS metrics have $\mathbb{N}^+$ as their value type; (c) QoS Profile subsumption reasoning used for matchmaking QoS specifications is quite slow; (d) no useful results are provided when all QoS offers do not match with the QoS demand.

Martín-Díaz et. al. [2] use a symmetric but syntactic QoS model and propose a CP-based approach for discovery. Before matchmaking, a QoS specification is transformed to a CSP [4] which is checked for *consistency/satisfiability*. Matchmaking is performed according to the concept of *conformance*. Concerning WS Selection, the (QoS) score of an offer is computed by a *Constraint Satisfaction Optimization Problem* (CSOP). This prominent approach has the following shortcomings: (a) it uses a syntactic QoS model; (b) it does not perform QoS metric matching; (c) it does not offer advanced categorization of results; (d) it does not offer useful results for over-constrained QoS demands.

3 QoS-Based Web Service Description

For a rich, semantic and extensible QoS-based WS description, we have developed an upper ontology called OWL-Q, which comprises of many sub-ontologies, each of which can be extended independently of the others. This ontology also complements OWL-S, the W3C submission for the functional specification of WSs. OWL-Q's design has now been finalized. In its new form, OWL-Q has eleven facets: main, directive, expression, function, measurement, metric, scale, spec, time, unit and value type. There are four most significant changes that have been made to OWL-Q. The first one is that now mathematical formulas and QoS specification guarantees are not expressed in OpenMath but in a subset of SWRL. Especially, guarantees are expressed in the form: $(f(arguments)|metric)$ *op value*, where f is a SWRL built-in, arguments are a list of built-ins, metrics and values, op can be one of $\leq, \geq, <, >, =, !=$. The second one is that measurements are now modeled so as to enable their storage and statistical processing by registries or other parties. Statistical processing leads to new metric derivation and to validation of QoS-based WS provider guarantees. The third one is modeling of scales. Scale is a more general concept than Unit. A Unit is actually a subclass of Ratio Scales. There are five disjoint subclasses of scales which actually control the operations on their metrics in functions or QoS goals.

The most significant change in OWL-Q is the incorporation of rules. It is well-known at the Semantic Web community that OWL supports very well reasoning about concepts but not about properties. For example, there is no way we can specify that a fact $p(x, y)$ can be true, where x, y are instances, if other

property or instance facts are true. As another example, there is no way to specify that two property facts cannot be both part of the semantic database. However, it is imperative in OWL-Q to reason about properties with rules because: **a)** relations between temporal properties like duration should be expressed and reasoned about; **b)** operations or comparisons on metrics should be restricted according to the scale that they use; **c)** integrity constraints between property facts or instance facts should be able to be enforced; **d)** compatibility or equivalency of scales and compatibility of metrics' value types should be expressed by OWL property facts fired by rules; **e)** rule-based algorithms like the metric matching one have to be specified. So we are currently in the process of extending OWL-Q with rules, which are expressed in SWRL – the currently promoted SW standard.

4 Constraint Programming

CP is the study of computational models and systems based on constraints. CP has been widely used in areas such as planning, scheduling and optimization. Currently, it is becoming the standard method for modeling optimization problems and is attracting widespread commercial interest as it is based on a strong theoretical foundation and can solve real-hard problems. In fact, it has been shown that *Logic Programming* (LP) is just a particular kind of CP.

A constraint is a relation among several variables, each of which ranges over a given domain. So a constraint restricts the values that its variables can take. In CP, a problem can be solved by first stating constraints about the problem area and, consequently, finding a solution that satisfies all the constraints. The last task is carried-out by a *solver*. Thus, CP uses constraints to declaratively state the problem without specifying a computational procedure to enforce them. Constraints in CP are generally expressed by a rich language that includes linear and non-linear constraints or logical combinations of constraints. Actually, the expressiveness of this language depends on the capabilities of the underlying solver.

A problem in CP expressed as a set of constraints is called CSP. A CSP is formalized as a set of variables V, a set of domain of values D and a set of constraints C. Domain of values can be reals, integers, boolean, enumerations or powersets and are associated to one or more variables. Constraints are mathematical expressions over a subset of V restricting the values these variables can take. A *solution* to a CSP is an assignment in which each variable in V takes a value from its corresponding domain in D as long as it does not violate the constraint set C. The *solution space* of a CSP is the set of all its solutions. A CSP is *satisfiable* if its solution space is not empty, i.e. it has at least one solution. For example, the CSP $(\{x, y\}, \{[0 \dots 2], [0 \dots 2]\}, \{x < y, x > 0\})$ is satisfiable as its solution space contains the sole solution $\{x \mapsto 1, y \mapsto 2\}$.

Instead of getting one or all solutions of a CSP, a user goal could be to find those solutions that satisfy an objective function. This type of CSP is called CSOP as it actually defines an optimization problem. In relation to CSPs, the

solution space of a CSOP is called *minimum space*. The objective function is just a function over a subset of V.

5 QoS-Based Web Service Discovery

5.1 QoS Metric Matching as a Non-linear CSP

All QoS-based WS discovery algorithms fail to produce accurate results because they rely on either syntactic or semantically-poor QoS metric descriptions. Hence, they cannot infer the equivalence of two QoS metrics based on descriptions provided by different parties. Different specifications of QoS metrics occur for two reasons: **a)** different perception of the same concept; **b)** different type of system reading for the same metric. For example, equivalent response time metrics could be associated to different units (e.g. minutes vs. seconds) and to different value types(e.g. [0.0,10.0] vs. [0,600] respectively). As another example, a DownTime metric can be either obtained in the form of high-level reading from a system with advanced instrumentation or can be derived from a resource metric of a system's Status obtained from low-level reading of systems with basic instrumentation.

Provided that two QoS metric descriptions are expressed in OWL-Q, we have developed a rule-based QoS metric matching algorithm [3] that infers the equivalence of the two metrics. This algorithm is composed of three main rules, each corresponding to a different case in a two metrics comparison. The last rule is recursive and reaches the final point of checking the equivalence of two mathematical formulas in order to infer the equivalence of two metrics. Unfortunately, equivalency of mathematical expressions is generally undecidable.

Due both to changes on the OWL-Q Ontology and to the above undecidability problem, we have modified our metric matching algorithm as follows:

$$\underline{match}\,(M_1, M_2) \Leftarrow \underline{rrm}\,(M_1, M_2) \vee \underline{rcm}\,(M_1, M_2) \vee \underline{ccm}\,(M_1, M_2) \tag{1}$$

$$sm\,(M_1, M_2) \Leftarrow svm\,(M_1.scale, M_2.scale, M_1.type, M_2.type)$$
$$\wedge\, M_1.object = M_2.object \wedge M_1.measures = M_2.measures \tag{2}$$

$$\underline{rrm}\,(M_1, M_2) \Leftarrow ResourceMetric\,(M_1) \wedge ResourceMetric\,(M_2) \wedge sm\,(M_1, M_2) \tag{3}$$

$$\underline{rcm}\,(M_1, M_2) \Leftarrow ResourceMetric\,(M_1) \wedge CompositeMetric\,(M_2) \wedge sm\,(M_1, M_2)$$
$$\wedge\, M_2.derivedFrom \cap CompositeMetric = \oslash$$
$$\wedge\, \neg\exists\, V \in M_2.derivedFrom\ match\,(M_1, V) \tag{4}$$

$$\underline{ccm}\,(M_1, M_2) \Leftarrow CompositeMetric\,(M_1) \wedge CompositeMetric\,(M_2)$$
$$\wedge\, sm\,(M_1, M_2) \wedge msm\,(M_1.derivedFrom, M_2.derivedFrom)$$
$$\wedge\, \neg solveCSP(M_1.derivedFrom, M_2.derivedFrom,$$
$$M_1.measuredBy - M_2.measuredBy! = 0) \tag{5}$$

where M_1 and M_2 are Metrics, $svm\,(M_1.scale, M_2.scale, M_1.type, M_2.type)$ is a rule that infers if the scales and value types of metrics M_1 and M_2 are compatible,

$msm\,(M_1.derivedFrom, M_2.derivedFrom)$ is a rule that matches one by one the M_1's list of derivative metrics with the corresponding metrics list of M_2, and $solveCSP\,(List_1, List_2, equation)$ is a logic procedure that solves the CSP defined by the two first metric lists and the equation given by third argument. When the latter procedure finds a solution, it returns true, otherwise it returns false. More details about all other clauses and symbols can be found in [3].

Therefore, the above algorithm infers that two metrics M_1 and M_2 match if one of the three body rules of rule (1) is satisfied. The first (rule (3)) of the three rules is satisfied if metrics M_1 and M_2 are resource metrics, if their scales and value types are compatible, if they measure the same service object (service, input, etc.) and if they measure the same QoS Property. Note that the last three clauses are defined by rule (2). The second (rule (4)) of the three rules is satisfied if M_1 is a resource and M_2 is a composite metric, if rule (2) is satisfied and if M_2 is derived only from resource metrics that are different from M_1 (i.e. they don't match with it).

The last of the three rules, rule (5), is satisfied if its five body clauses are satisfied. The first two clauses check if metrics M_1 and M_2 are composite ones. The third clause is actually rule (2). The fourth clause compares one by one the metrics from which M_1 is derived with the corresponding metrics of M_2. If two metrics are matched, then they are replaced by a new metric in the derivation lists of M_1 and M_2. In addition, this replacement also takes place at the measurement formulas of M_1 and M_2. Lastly, if all previous clauses are satisfied, then from the derivation lists of M_1 and M_2 and their measurement formulas a (possibly non-linear) CSP is defined and solved, based on the following transformation:

1. $\forall M_i \in M_1.derivedFrom \notin M_2.derivedFrom$ map M_i to CSP variable $X_{i,1}$ and assign the value type of M_i to the value domain of this variable
2. $\forall M_j \in M_2.derivedFrom \notin M_1.derivedFrom$ map M_j to CSP variable $X_{j,2}$ and assign the value type of M_j to the value domain of this variable
3. $\forall M_k \in M_1.derivedFrom \cap M_2.derivedFrom$ map M_k to CSP variable $X_{k,12}$ and assign the value type of M_k to the value domain of this variable
4. Add the constraint $f\,(X_{i,1}, X_{k,12}) - g\,(X_{j,2}, X_{k,12})! = 0$ to the CSP, where $f = M_1.measuredBy$, $g = M_2.measuredBy$.

If the CSP is unsatisfiable i.e. it hasn't any solution at all, then the last clause is also satisfied and rule (5) is satisfied.

Composite-to-Composite Metric Matching Example. Assume that a WS provider defines composite metric $Avail_1$ that measures the QoS Property of *Availability* of his whole WS and is derived from two Resource metrics $Downtime_1$ and $Uptime_1$ based on the formula: $1 - Downtime_1/(Downtime_1 + Uptime_1)$. In addition, assume that a WS requester defines composite metric $Avail_2$ that also measures the QoS Property of *Availability* and is derived from two Composite metrics $Downtime_2$ and $Uptime_2$ based on the formula: $Uptime_2/(Uptime_2 + Downtime_2)$. Further assume that all metrics have as value type the interval $[0.0, 1.0]$ and that the following facts are true:

$sm(Avail_1, Avail_2)$, $rcm(Downtime_1, Downtime_2)$, $rcm(Uptime_1, Uptime_2)$. We want to see if composite metrics $Avail_1$ and $Avail_2$ are matched based on the satisfiability of rule (5).

The first three clauses of this rule are trivially true. Based on the facts: $rcm(Downtime_1, Downtime_2)$ and $rcm(Uptime_1, Uptime_2)$, the procedure represented by the fourth clause will map $Downtime_1$ and $Downtime_2$ to a new metric $Downtime$. Similarly, it will map $Uptime_1$ and $Uptime_2$ to $Uptime$. In this way, it stands that: $Avail_1.derivedFrom = Avail_2.derivedFrom = [Downtime, Uptime]$, $Avail_1.measuredBy = 1 - Downtime/(Downtime + Uptime)$, $Avail_2.measuredBy = Uptime/(Uptime + Downtime)$. The procedure represented by the last clause of the rule will create and solve a CSP having the following definitions: $Downtime, Uptime :: [0.0, 1.0]$ and constraints: $1 - Downtime/(Downtime + Uptime) - Uptime/(Uptime + Downtime)! = 0$. As this CSP is unsatisfiable, the facts $ccm\,(Avail_1, Avail_2)$ and $match\,(Avail_1, Avail_2)$ are inferred. Thus, by the last inferred fact, we have concluded that QoS metrics $Avail_1$ and $Avail2$ are matched.

5.2 QoS-Based Web Service Discovery Algorithm

One of the most prominent QoS-based WS discovery algorithm [2] expresses each QoS-based WS description as a CSP. Then it separates the QoS-based advertisements into two categories: the ones that satisfy completely the QoS-based request and the others that do not satisfy the request. However, this algorithm presents four major drawbacks: **1)** It provides a syntactic QoS (metric) model. **2)** It does not perform QoS metric matching. **3)** It does not provide advanced categorization of matches. **4)** It does not deal with cases where all QoS offers do not conform to the (over-constrained) demand.

The first two drawbacks lead to matchmaking results with low precision and recall. The third drawback lies on the fact that the algorithm under investigation provides only two useful categories of results: *matched* and *failed* QoS offers. However, the WS requester would like to find out more interesting features of the results. For example, he would like to know which matched QoS offers are more constrained than the QoS demand. The last drawback is actually a consequence of the third drawback. When an over-constrained QoS demand is issued, this algorithm only returns failed matches. This is not quite useful. First of all, the WS requester does not know which QoS offers violate all the constraints of the QoS demand and which not. It would be quite useful if failed matches were further categorized into two subcategories indicating the amount of failure (complete or not). Secondly, it would be quite useful if the user was informed that he could obtain matched QoS offers by relaxing the least number of his constraints.

In the sequel, we present an automated QoS-based WS matchmaking algorithm that exploits OWL-Q and the QoS metric matching algorithm in order to extend the aforementioned algorithm. This algorithm takes as input the QoS offers of all the WS advertisements and the QoS demand of the request in the form of OWL-Q specifications and returns four ordered lists of WS advertisements. It is more complete than the one [3] we have presented in a previous

conference. It is composed of four parts/processes, which are analyzed in the four following subsections. Finally, the last subsection provides a small example of its application into a set of offers and one demand.

Alignment Process. Before analyzing our discovery algorithm, let us clarify its input. The input consists of I OWL-Q offers O_i $\left(1 \leq i \leq I, I \in \mathbb{N}^+\right)$ and one OWL-Q demand D. Every offer O_i guarantees a conjunctive list of goals G_{ij} with $1 \leq j \leq J_i$ and $J_i \in \mathbb{N}^+$. Every goal G_{ij} references K_{ij} metrics $M_{ij\,k}$, where $1 \leq k \leq K_{ij}$ and $K_{ij} \in \mathbb{N}^+$. Similarly, the demand D guarantees N conjunctive goals G_n^D, where $1 \leq n \leq N$ and $N \in \mathbb{N}^+$. Each goal G_n^D may have a weight $w_n^D \in [0.0, 1.0) \cup \{2.0\}$ and references L_n metrics $M_{n\,l}^D$, where $1 \leq l \leq L_n$ and $L_n \in \mathbb{N}^+$. Goals actually correspond to constraints, so if a demand's constraint has weight w in $[0.0, 1.0)$, then it is considered a *soft* constraint, otherwise it is a *hard* constraint that must be satisfied at any cost. Demand D also references a selection list S of the form (metric,weight).

The Alignment process of our discovery algorithm aligns all offers O_i and demand D by finding their common QoS metrics by exploiting the QoS metric matching algorithm. It returns their corresponding CSPs, after it has validated that these CSPs are consistent (i.e. they have a solution). If it finds an inconsistent offer CSP, it removes this offer from the discovery algorithm. If the demand's CSP is inconsistent, then the whole discovery algorithm fails. However, the CSPs of the offers are stored. The alignment process relies on the concept of the Metric Store (MS). The MS stores all unique QoS metrics encountered so far. So when a new QoS spec arrives, we don't need to examine if any of its metrics matches with any metric of all offers or demands but with any metric in the MS. In this way, there is a minimization of all the possible metric-to-metric comparisons. In case of a process failure, the MS's content also stays intact.

The process starts by examining the first QoS offer. It adds all of the offer's unique metrics to the MS and then it transforms it to the corresponding CSP. It solves the CSP and if it is inconsistent, then it adds the offer to a delete list. The process treats all offers similarly. However, when comparing an offer's metric with the MS, if there is a match, then a List of (offer's metric, MS metric) is updated with a corresponding entry. Otherwise, the offer's metric is added to the MS. After all offers are processed, all inconsistent offers are deleted. Then the process works out the QoS demand similarly with the offers with the exception that if its CSP is inconsistent, then the whole discovery algorithm fails.

The transformation of a QoS spec to a CSP is carried away as follows: Initially, the CSP is empty. Then, for every unique metric of the QoS spec, we take two steps: **a)** We check if it was matched or not; If yes, then we get the matching MS metric and its position j in the MS and we add a definition to the CSP: $X_j :: a..b$, where $[a, b]$ is the value range of the matching MS metric. Otherwise, we get the position of the new metric in the MS and we add the definition: $X_j :: a..b$, where $[a, b]$ is the value range of the metric; **b)** For every goal, we check if the metric is contained in its expression. If yes, then if it was new, we update the goal's expression by substituting the name of the metric with the variable X_j; if it was matched, then we update the goal's expression by first

substituting the name of the metric with the variable X_j and then applying the scale-to-scale transformation function to X_j in order not to change the meaning of the goal/constraint. After all metrics have been processed, for every goal we add its modified expression as a constraint to the CSP. For example, if the matching MS metric has scale Minute and domain [0.0,2.0], the spec's metric has scale Second and domain [1,120] and has goal $Metric >= 100$, then we add to the CSP the definition $X_j :: 0.0..2.0$ and the constraint $60 * X_j >= 100$.

Matchmaking Process. This process is based on the concept of conformance [2], which is mathematically expressed by the following equivalency:

$$conformance\,(O_i, D) \Leftrightarrow sat\left(P_i^O \wedge \neg P^D\right) = \text{false} \tag{6}$$

where *sat* is a procedure inferring the satisfiability of a logical combination of CSPs. To explain, an offer O_i matches a demand D when there is no solution to the offer's CSP P_i^O that is not part of the solution set of the demand's CSP P^D. As a CSP is a conjunction of constraints, we can rewrite (6) as:

$$\begin{aligned}
&conformance\,(O_i, D) \Leftrightarrow sat\left(P_i^O \wedge \neg P^D\right) = \text{false}\\
&\Leftrightarrow sat\left(P_i^O \wedge \neg\left(con_1^D \wedge con_2^D \wedge \cdots \wedge con_m^D\right)\right) = \text{false}\\
&\Leftrightarrow sat\left(P_i^O \wedge \left(\neg con_1^D \vee \neg con_2^D \vee \cdots \vee \neg con_m^D\right)\right) = \text{false}\\
&\Leftrightarrow sat\left(\left(P_i^O \wedge \neg con_1^D\right) \vee \left(P_i^O \wedge \neg con_2^D\right) \vee \cdots \vee \left(P_i^O \wedge \neg con_m^D\right)\right) = \text{false}\\
&\Leftrightarrow sat\left(P_i^O \wedge \neg con_1^D\right) = sat\left(P_i^O \wedge \neg con_2^D\right) = \cdots = sat\left(P_i^O \wedge \neg con_m^D\right) = \text{false}
\end{aligned}$$

So, an offer O_i conforms to the demand D, if and only if all CSP problems – constructed by offer's CSP P_i^O, the negation of a demand's constraint $\neg con_j^D$ and the definitions of variables participating in the negated constraint and not defined at the offer's CSP – are unsatisfiable.

This matchmaking process takes as input the CSPs of the offers and the demand and produces four types of results. For each offer's CSP P_i^O it checks if it conforms to the demand's CSP P^D by constructing N CSP problems (where N is the number of the constraints of the demand) and solving all of them. If all CSPs are unsatisfiable then there are two cases: **a)** if the offer's CSP contains constraints on variables not included at the CSP of the demand, then the offer and its CSP P_i^O are put at the *super match* list; **b)** otherwise the offer and its CSP P_i^O are put at the *exact match* list. For every CSP (from the N) that a solution is found, a counter is increased, the weight of the demand's constraint is added to a second counter and the demand's non-negated constraint is put into a list. If at the end of the problem solving the counter equals N, then the offer and its CSP P_i^O are put at the *fail match* list. Otherwise, the offer, its CSP P_i^O, the value of the two counters and the demand's violating constraints list are put at the *partial match* list as an entry. The first counter counts the number of satisfiable CSPs (i.e. number of conflicting original constraints of the demand) and the second counter measures the total sum of weights of these conflicting constraints. These constraints of the demand's CSP P^D do not allow

some solutions of the offer's CSP P_i^O to be part of the demand's solution space. If they get relaxed, then the corresponding offer will conform to the demand.

Thus, the matchmaking process produces four types of results: *super*, *exact*, *partial*, *fail* with decreasing order of significance. *Super* offers not only conform to the demand but also impose constraints on metrics not referenced by the demand's constraints. *Exact* offers just conform to the demand by using the same metrics. These two result types represent offers which conform to the demand. *Partial* offers do not conform to the demand because either they do not use some metrics of the demand or they contain solutions that are not part of the demand's solution space. So *partial* results are promising, especially if the first two lists of results are empty. *Fail* offers contain lower quality solutions with respect to the solutions requested by the demand. If only *fail* matches are produced, then this is an indication of an over-constrained demand. In this case, the discovery algorithm fails and an appropriate warning is issued to the WS requester.

Constraint Relaxation Process. Assume that the matchmaking process returns only *partial* and *fail* type of results. *Fail match* results are of no use and are not further processed. However, *partial* results are promising as they represent QoS offers that don't use QoS metrics of the demand or have solutions that are not included in the solution space of the demand's CSP. If the first case holds, then two alternative solutions are possible. The first solution is to find the offer(s) with the smallest set of same undefined variables/metrics and then continue to the next process to order these offers. The user gets back the ordered list and an indicating message that his query was relaxed by removing some metrics and their unary constraints. However, this solution is not preferable if these removed variables participate in n-ary constraints ($n \geq 2$) involving not-removed variables as these constraints will also have to be removed.

The second solution of the first case is better and more general. Instead of removing an offer's CSP variables, we ask its service provider to enrich it with definitions of missing metrics and possible constraints on them. Then we restart the discovery algorithm with only these new offers and the original demand.

For the second case, the matchmaking process has provided two metrics for each *partial* offer: number of demand's conflicting original constraints and their total weight. So we order the *partial* list of offers according first to the total weight and then to the number of the conflicting constraints. In this way, at the top will be offers having the smallest number of hard constraints and the least total weight of weak constraints. So we put these topmost offers along with their conflicting constraints at the *exact match* list and we move to the last process in order to rank them and return them. However, the user is warned that this ordered list represents *exact* matches only if the user weakens the corresponding conflicting constraint list from his demand.

Selection Process. The goal of the selection process is to rank the best result lists produced by the previous processes. If *super* and/or *exact* matches exist, then they are ranked. Otherwise, there will be only *partial* matches to be ranked. By providing a sorted list of the best possible matches, the WS requester is

supported in choosing the best QoS offer according to his preferences, which are expressed by a list associating weights to QoS metrics. Our selection process [3] extends the one defined in [2] by **a)** having semantically aligned CSPs of offers; **b)** the score of each offer is produced by the weighted sum of the score of its worst solution plus the score of its best solution by having two CSOPs solved. More details can be found in [3].

Complete Example. To demonstrate our QoS-based WS discovery algorithm, we supply a simple example of its application to a small set of three QoS offers O_i and one demand D. The first offer O_1 references three metrics: **a)** $Resp1$ that measures the QoS property of Response Time, uses the scale of seconds and has $(0.0, 86400]$ as value type; **b)** $Thr1$ that measures the QoS property of Throughput, uses the scale of requests per second and has $(0, 100000]$ as value type; **c)** $Avail1$ that measures the QoS property of Availability, uses the scale of ratio and has $(0.0, 1.0)$ as value type; and has the following list of goals: $Resp1 \leq 10.0 \wedge Thr1 \leq 100 \wedge Thr1 \geq 50 \wedge Avail1 \geq 0.9$. The second offer O_2 uses three metrics: **a)** $Resp2$ measuring Response Time, using the scale of minutes and having $(0.0, 1440.0]$ as value type; **b)** $Thr1$; **c)** Avail1; and has the following list of goals: $Resp2 \leq 0.08 \wedge Thr1 \leq 50 \wedge Thr1 \geq 40 \wedge Avail1 \geq 0.95$. The third offer O_3 uses three metrics: **a)** $Resp2$; **b)** $Thr1$; **c)** $Avail2$ measuring Availability, using the scale of percentage and having $(0.0, 100.0)$ as value type; and has the following goals: $Resp2 \leq 0.06 \wedge Thr1 \leq 40 \wedge Thr1 \geq 30 \wedge Avail2 \geq 98.0$. Finally, demand D uses the metrics: **a)** $Resp1$; **b)** $Thr1$; **c)** $Avail2$ and has the following goals: $Resp1 \leq 15.0 \wedge Thr1 \geq 40 \wedge Avail2 \geq 99.0$. Moreover, the WS requester does not provide weights to the constraints of his demand and associates the following weights to his defined three metrics: $Resp1 \leftarrow 0.3$, $Thr1 \leftarrow 0.3$, $Avail2 \leftarrow 0.4$, while $a = 0.7$ and $b = 0.3$ [3].

We now apply the four processes of our discovery algorithm. The alignment process matches metrics of the offers and the demand, stores first-seen QoS metrics at the MS and produces four CSPs P_i and P^D. All CSPs have the following three definitions: $X_1 :: (0.0, 86400.0]$, $X_2 :: (0, 100000]$ and $X_3 :: (0.0, 1.0)$. Based on these variable definitions, each CSP has the following constraints: $P_1 : X_1 \leq 10.0 \wedge X_2 \leq 100 \wedge X_2 \geq 50 \wedge X_3 \geq 0.9$, $P_2 : X_1/60.0 \leq 0.08 \wedge X_2 \leq 50 \wedge X_2 \geq 40 \wedge X_3 \geq 0.95$, $P_3 : X_1/60.0 \leq 0.06 \wedge X_2 \leq 40 \wedge X_2 \geq 30 \wedge 100 \cdot X_3 \geq 98.0$ and $P^D : X_1 \leq 15.0 \wedge X_2 \geq 40 \wedge 100 \cdot X_3 \geq 99.0$. The matchmaking process checks the satisfiability of 9 CSPs (3 offer CSPs times 3 demand constraints) and produces four lists of results: $Super = [\,]$, $Exact = [\,]$, $Partial = [(O_1, P_1, 1, 2.0, 100 \cdot X_3 \geq 99.0), (O_2, P_2, 1, 2.0, 100 \cdot X_3 \geq 99.0), (O_3, P_3, 2, 4.0, [X_2 \geq 40, 100 \cdot X_3 \geq 99.0])]$, $Fail = [\,]$. The constraint relaxation process sorts the partial list based on the third and fourth argument of each entry and produces two lists that are passed to the last process. These lists are: $Exact^* = [[(O_1, P_1), (O_2, P_2)], 100 \cdot X_3 \geq 99.0]$ and $Partial = [O_3, P_3]$. Finally, the selection process solves two CSOPs for each of the first two offers, produces one score for each: $Score_1^O \simeq 0.37$ and $Score_2^O \simeq 0.38$, and returns the following ordered list to the requester: $Matches = [(O_2, 0.38), (O_1, 0.37)]$. However, the WS requester is warned that this list contains offers that do not satisfy his constraint: $100 \cdot X_3 \geq 99$.

As it can be seen from this example, offer O_2 is at the top of the result list as it violates in a significantly lower amount the last constraint of the demand with respect to the amount of violation of O_1. So if O_2 is selected by the WS requester, then the minimum possible constraint relaxation would have been achieved.

6 QoS-Based Web Service Discovery Engine

We are currently in the development phase of our QoS-based WS discovery engine by using the Pellet reasoner (`http://http://pellet.owldl.com`) for ontology reasoning and the ECLiPSe (`http://eclipse.crosscoreop.com`) system for solving linear constraints. Pellet is chosen because it is one of the best three ontology reasoners supporting the tasks of ontology validation and reasoning, OWL 1.1 datatype reasoning and partial SWRL inferencing. ECLiPSe is chosen as it is one of the most efficient Constraint Logic Programming languages that supports advanced linear constraint solving and extends the common facilities of Prolog. Moreover, it can be extended to support non-linear constraint solving through external solvers. The architecture of the system under development is shown in Fig. 1. In the sequel, an overview of the functionality of each component of our system is provided by analyzing the scenario where a WS provider wants to publish the QoS offer of his WS while a WS requester wants to find available QoS offers based on his request.

Publication. The *WS Provider* (WSP) describes his QoS offer in OWL-Q and sends it to the *Java Server*, the main component of our system. The *Java Server* (JS) sends this offer to the *Reasoner* (RS) in order to validate its (ontological) consistency against the common OWL rules and the accompanying SWRL rules. If the offer is not consistent, then an error message is returned to the WSP. Otherwise, JS aligns the OWL-Q offer according to the contents of the *Metric*

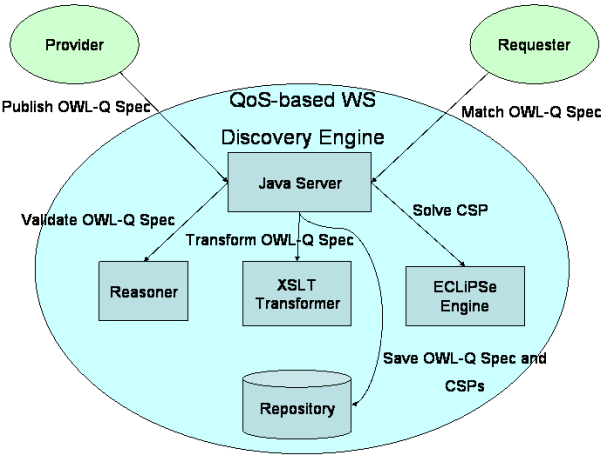

Fig. 1. QoS-based WS Discovery Engine

Store (MS), which is part of the *Repository* (R), and with the help of the *XSLT Transformer* (XSLT) it transforms it into an ECLiPSe CSP. This CSP is send to the *ECLiPSe Engine* (EE) in order to find if it is satisfiable. If not, then an appropriate error message is returned to the WSP. If yes, then the OWL-Q offer and its accompanying CSP are stored at R and an appropriate positive message is returned to the WSP. Note that the alignment process also produces CSPs, which are also forwarded to the EE.

Matching. The *WS Requester* (WSR) sends his OWL-Q demand to the JS, which follows exactly the same procedure as above. If everything is alright, then the demand and its CSP are stored at R for caching purposes. Then the remaining three processes of matchmaking, constraint relaxation and selection process are executed at JS, which suspends execution to send appropriate CS(O)Ps for solving to the EE and resumes when it gets the results back. In the end, the user gets an ordered list of matching OWL-Q offers or a list of partial matches along with a suggestion or a matching failure message.

7 Future Work

As future work, we plan to modify our constraint relaxation algorithm so as to exploit advanced techniques for solving over-constrained problems like semiring based constraint satisfaction [12]. In addition, we have identified that the metric of conformance is not perfect as it excludes from the discovery result set offers that provide better solutions than that of the demand. So we plan to change the mathematical definition of conformance appropriately and alter our matchmaking algorithm accordingly. We also plan to extend OWL-Q with the description of the context of both the WS and its requester. We believe that a Context-aware WS discovery process will be more accurate and customizable as the tasks of request and input completion, output adaptation and added-value composition of service-offerings become possible. Our ultimate and final goal is to achieve QoS-based and context-aware WS composition.

References

1. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: AAMAS 2006: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, pp. 915–922. ACM Press, New York (2006)
2. Cortés, A.R., Martín-Díaz, O., Toro, A.D., Toro, M.: Improving the automatic procurement of web services using constraint programming. Int. J. Cooperative Inf. Syst. 14(4), 439–468 (2005)
3. Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: ECOWS 2006: Proceedings of the European Conference on Web Services, Washington, DC, USA, pp. 265–274. IEEE Computer Society, Los Alamitos (2006)
4. Van Hentenryck, P., Saraswat, V.: Strategic directions in constraint programming. ACM Computing Surveys 28(4), 701–726 (1996)

5. Ran, S.: A model for web services discovery with qos. SIGecom Exch. 4(1), 1–10 (2003)
6. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Rec. 31(4), 36–41 (2002)
7. Tosic, V., Pagurek, B., Patel, K.: Wsol - a language for the formal specification of classes of service for web services. In: Zhang, L.J. (ed.) ICWS, pp. 375–381. CSREA Press (2003)
8. Keller, A., Ludwig, H.: The wsla framework: Specifying and monitoring service level agreements for web services. Technical Report RC22456 (W0205-171), IBM (2002)
9. Tian, M., Gramm, A., Nabulsi, M., Ritter, H., Schiller, J., Voigt, T.: Qos integration in web services. Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML (October 2003)
10. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic ws-agreement partner selection. In: WWW 2006: Proceedings of the 15th international conference on World Wide Web, pp. 697–706. ACM Press, New York (2006)
11. Zhou, C., Chia, L.T., Lee, B.S.: Daml-qos ontology for web services. In: ICWS 2004: Proceedings of the IEEE International Conference on Web Services (ICWS 2004), Washington, DC, USA, p. 472. IEEE Computer Society, Los Alamitos (2004)
12. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM 44(2), 201–236 (1997)