Semantic QoS-based Web Service Discovery Algorithms

Kyriakos Kritikos and Dimitris Plexousakis
Foundation for Research and Technology Hellas, Heraklion, Greece
kritikos@ics.forth.gr and dp@ics.forth.gr

Abstract

The success of the Web Service (WS) paradigm has led to a proliferation of available WSs, which are advertised in WS registries. While sophisticated semantic WS discovery algorithms are operating on these registries to return matchmaking results with high precision and recall, many functionally-equivalent WSs are returned. The solution to the above problem 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 called OWL-Q. We have also proposed a semantic QoS metric matching algorithm. Based on this algorithm, we have extended a CSP-based approach for QoS-based WS discovery. In this paper, we firstly analyze the evolution of OWL-Q and its extension with SWRL rules, we propose a modification to the metric matching algorithm and we show the way the metric alignment process takes place. Then we propose two novel semantic QoS-based WS Discovery algorithms that return matches even for over-constrained QoS-based WS requests. The first one deals with unary constraints while the second one is more generic. Finally, implementation aspects of our QoS-based WS discovery system are discussed.

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, like the one in [3], have been invented to overcome the above 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 or just one value. Actually, the current modeling efforts of QoS specifications only differ in the expressiveness of these constraints. However, these efforts fail in QoS 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, like the one in [5], produce irrelevant or incomplete results.

Based on the above deficiencies, we have developed OWL-Q [4], 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 [5]. 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. Following, a small introduction in *Constraint Programming* (CP) [11] will be provided, emphasizing on the definitions of notions like *Constraint*

Satisfaction Problem (CSP) and *Constraint Satisfaction Optimization Problem* (CSOP). Next, we clarify the way the QoS metric matching algorithm has been modified to compute the satisfiability of possibly non-linear CSPs instead of inferring mathematical equivalency of formulas. In this way, this algorithm becomes computationally feasible. In addition, we show the way this algorithm can be exploited to align QoS-based WS specifications and transform them to CSPs. Then, we propose two CSP-based QoS-based WS Discovery algorithms that provide solutions even for over-constrained QoS-based WS demands. The first one deals only with unary constraints and is very simple and quick. The second is more generic as it deals with arbitrary constraints and uses *Constraint Satisfaction Optimization* (CSO) techniques. Next, we analyze the architecture and functionality of our QoS-based WS Discovery System, 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. While *OWL-S* is a standard *semantic* approach for WS Description, it does not describe any QoS concept.

Ran [8] 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. [10] 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) [2] 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. [7] offer a semantic framework for the definition and matching of *WS-Agreements*. However, only unary QoS metric constraints can be expressed while QoS metric matching could only be enforced by manual incorporation of rules.

Zhou et. al. [12] 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 range; (c) QoS Profile subsumption reasoning is quite slow.

Martín-Díaz et. al. [5] use a symmetric but syntactic QoS model and propose a CSP-based approach for discovery. Before matchmaking, a QoS specification is transformed to a CSP 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 solving a *Constraint Satisfaction Optimization Problem* (CSOP).

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 [4], 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 semantic 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 WS Specification Alignment and Transformation

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 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 [4] 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 – used in complex-to-complex metric matching – 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 and not really effective in pragmatic circumstances.

Due both to changes on the OWL-Q Ontology and to the above undecidability problem, we have modified

our metric matching algorithm as follows:

$$mm\,(M_1, M_2) \Leftarrow rrm\,(M_1, M_2) \vee rcm\,(M_1, M_2) \vee ccm\,(M_1, M_2) \quad (1)$$

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

$$rrm\,(M_1, M_2) \Leftarrow RM\,(M_1) \wedge RM\,(M_2) \wedge sm\,(M_1, M_2) \quad (3)$$

$$rcm\,(M_1, M_2) \Leftarrow RM\,(M_1) \wedge CM\,(M_2) \wedge sm\,(M_1, M_2) \wedge M_2.derivedFrom \cap CompositeMetric = \oslash \wedge \neg\exists\, V \in M_2.derivedFrom\; match\,(M_1, V) \quad (4)$$

$$ccm\,(M_1, M_2) \Leftarrow CM\,(M_1) \wedge CM\,(M_2) \wedge sm\,(M_1, M_2) \wedge msm\,(M_1.derivedFrom, M_2.derivedFrom) \wedge \neg solveCSP(M_1.derivedFrom, M_2.derivedFrom, M_1.measuredBy - M_2.measuredBy! = 0) \quad (5)$$

where M_1 and M_2 are Metrics, $RM\,(M)$ and $CM\,(M)$ are rules inferring if metric M is resource and composite respectively, $svm\,(M_1.scale, M_2.scale, M_1.type, M_2.type)$ 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 [4].

In relation to our previous matching algorithm [4], we have modified the third main rule (rule (5)) so as to reach the final point of defining and solving a (possibly non-linear) CSP in order to infer the matching of two complex metrics. Suppose that all previous body clauses of our rule are satisfied. In addition, suppose that all possible matches between the metrics – from which our two compared metrics are derived – have been inferred. Then, from the derivation lists $M_1.derivedFrom$ and $M_2.derivedFrom$ of the compared metrics M_1 and M_2 and their corresponding measurement formulas $M_1.measuredBy$ and $M_2.measuredBy$, a CSP is produced by 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 of our rule is satisfied. In result, the head of the rule is fired and the match between the two compared metrics is derived. More details about the algorithm can be found in [4]. Following, a simple example of composite-to-composite metric matching is provided for illustration purposes.

5.1.1 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 WS and is derived from two Resource metrics $Downtime_1$ and $Uptime_1$ based on the formula: $1 - Donwtime_1/(Downtime_1 + Uptime_1)$. 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 two compared metrics use the same unit. Finally, assume that the following matches are true: $match(Downtime_1, Downtime_2)$, $match(Uptime_1, Uptime_2)$ and that all previous clauses of the third main rule are satisfied. We want to see if composite metrics $Avail_1$ and $Avail_2$ are matched based on the satisfiability of the last clause of the rule. Based on the aforementioned transformation procedure, a CSP is created and solved that has the following definitions: $D, U :: [0.0, 1.0]$ and constraints: $1 - D/(D + U) - U/(U + D)! = 0$. This CSP is unsatisfiable so the last clause is satisfiable and finally the fact $match\,(Avail_1, Avail_2)$ is inferred.

5.2 Alignment and Transformation

The Alignment process is executed when any QoS specification S is published or queried on the underlying QoS-based WS discovery system. Its goal is to align S with all already processed offers O_i and demands D_j by finding their common QoS metrics based on the QoS metric matching algorithm. After metric

alignment, S is transformed to a CSP which is checked for consistency (i.e. if it has a solution). If the CSP is inconsistent, then neither S nor its CSP are stored in our *Repository* (R) and S's owner is informed. In case of an inconsistent demand, the discovery algorithm is also not executed. The alignment process relies on the concept of the *Metric Store* (MS), which is part of R. 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 possible metric-to-metric comparisons. In addition, all unique metrics of this new QoS spec are added to the MS. If this QoS spec is inconsistent, its metrics are not removed from the MS.

The transformation of a QoS spec S to a CSP is carried away as follows: Initially, the CSP is empty. Then, for every unique metric M of S, we take two steps: **a)** We check if it was matched or not; If yes, then we get the matching MS metric M' 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 M'. Otherwise, we get the position of M in the MS and we add the definition: $X_j :: a..b$, where $[a, b]$ is the value range of M; **b)** For every goal G of S, we check if M is contained in its expression. If yes, then if M was new, we update G's expression by substituting the name of M with the variable X_j; if it was matched, then we update G's expression by first substituting the name of M 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 G. 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 $60 * X_j >= 100$.

6 QoS-based Web Service Discovery Algorithms

One of the most prominent QoS-based WS discovery algorithm [5] 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 three major drawbacks: **1)** it performs syntactic metric matchmaking producing false negative and false positive results; **2)** QoS spec matchmaking relies on the concept of *conformance*, which is not absolutely correct (see next paragraph); **3)** it does not provide advanced categorization of results.

Matchmaking of QoS offers and demands is based on the concept of conformance [5], which is mathematically expressed by the following equivalency:

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

To explain, an offer O_i matches a demand D when there is no solution to the offer's CSP P_i that is not part of the solution set of the demand's CSP P^D. This definition is slightly wrong as it excludes from the result set those QoS offers that provide better solutions than that of the demand's. For example, suppose that a WS provider and requester use the same metric X, measuring the QoS Property of Availability, that has as value type the set $(0.0, 1.0)$. Further assume that the WS provider's CSP has the constraint: $X \geq 0.96$ while the WS requester's CSP has the constraint: $0.95 \leq X \leq 0.999$. Based on the above definition, the provider's offer does not match the request as it contains solutions greater than that of the request's, although these solutions are better. Thus, a more correct definition of matchmaking is the following: an offer O_i matches a demand D when its CSP P_i has solutions that are either contained in the solution set of the demand's CSP P^D or are better that the demand's solutions.

Based on the deficiencies of [5] and the new definition of matchmaking, we propose two QoS-based WS discovery algorithms that are analyzed in the following subsections. The first one is only restricted to unary constraints but is more effective and easy to implement while the other is more generic but harder to implement. These algorithms presuppose that the offers set $\{O_i\}$ and the demand D are already aligned and transformed to corresponding CSPs P_i and P^D respectively.

6.1 Unary Constraints Discovery Algorithm

Suppose we have N QoS offers O_i that are transformed to N CSPs P_i. Each P_i has constraints C_j^i of the form X op v, where X is a variable, op is one of $\{<, >, \leq, \geq, !=, =\}$ and v is a single value. In addition, we have a QoS demand D that is transformed to a CSP P^D which has constraints of the previous form. Further, suppose that each variable X has as value type the set $[a, b]$, where $a, b \in \mathbb{R}$ or $\mathbb{N}$ and $a \leq b$. Moreover, suppose that each constraint C of P^D has a weight $w_C \in (0.0, 1.0) \cup \{2.0\}$. A weight of 2.0 indicates that the constraint is *hard* and must be satisfied at any cost. A weight in $\{(0.0, 1.0)\}$ indicates that

the constraint is *soft*. Higher-weighted soft constraints should be satisfied first with respect to lower-weighted soft constraints.

Our first QoS-based WS Discovery algorithm takes as input the CSPs P_i and P^D of the QoS offers and demands respectively and produces four ordered categories of results. As it is based on unary constraints, it is easy to implement in any programming language and does not require the use of CSP engines. In result, it is easy to show that it will have very good performance. Obviously, the price we pay for simplicity and enhanced performance is reduced constraint expressivity. This algorithm consists of four sequential processes, which are analyzed in the following paragraphs.

CSP Preprocessing. The *CSP Preprocessing* process has two sequential goals: **1)** minimization of offers having missing variables/metrics; **2)** minimization of the number of constraints of all CSPs. So this process produces new CSPs P_i' and P'^D of each offer O_i and the demand D respectively. The **first** goal is achieved by asking the WS providers, having offers whose CSP does not have unary constraints on metrics/variables that are contained in constraints at the CSP of the demand, to enrich them with corresponding constraints on these metrics/variables. The **second** goal is achieved by manipulating for each CSP (of QoS offers and the demand) the following cases:

- Identical constraints are all removed except from one.
- Constraints $X \leq v$ and $X = v$ are replaced with $X = v$. In addition, constraints $X \geq v$ and $X = v$ are replaced with $X = v$.
- Constraints $X \leq v$ and $X! = v$ are replaced with $X < v$. In addition, constraints $X \geq v$ and $X! = v$ are replaced with $X > v$.
- Constraints of the form: Xop_1v, where $op_1 = \{>, \geq\}$, are all removed except from the one that has the greatest v. If there are two with the greatest v then the constraint with $op_1 = \geq$ is removed.
- Constraints of the form: $Xop_2\,v$, where $op_2 = \{<, \leq\}$, are all removed except from the one that has the smallest v. If there are two with the smallest v then the constraint with $op_1 = \leq$ is removed.
- If $Xop_1\,a$ or $Xop_2\,c$ and $X = b$ are present, where $op_1 \in \{>, \geq\}, op_2 \in \{<, \leq\}$ and $a < b < c$, then the inequality constraint(s) are removed.
- If $Xop_1\,a$ or $Xop_2\,c$ and $X! = b$ are present, where $op_1 \in \{>, \geq\}, op_2 \in \{<, \leq\}$ and $a < c, b < a$ or $b > c$, the difference constraint is removed.

Matchmaking The previous process has produced offers and demand CSPs that contain one of the following forms of constraints for each variable X: $[X = v]$, $[X! = v_1, X! = v_2, \ldots, X! = v_n]$, $[Xop_1\,a, Xop_2\,b]$, where $v, v_1, v_2, \ldots, v_n, v, a, b \in domain\,(X), a < b, op_1 = \{>, \geq\}$ and $op_2 = \{<, \leq\}$. The first two forms are usually used for variables that have small sets of integers as value types. The third form (where we can have one or both of the two constraints) is used for all other cases. Let us now analyze the two different cases of matchmaking CSPs P_i and P^D of one offer O_i and one demand D respectively (according to the domain of a common variable of their CSPs).

If the domain of the common variable X is a small set of integers, then there are four cases to consider:

1. P_i has the constraint $X = v$ and P^D has the constraints $X! = v_1, X! = v_2, \ldots, X! = v_n$. In this case, P^D's set of constraints are violated if and only if: $worse(v, worseOf(domain(X) - \{v_1, v_2, \ldots, v_n\}))$. Function $worse(v_1, v_2)$ returns true if v_1 is a worse value than v_2 according to the domain of X and the value monotonicity of the metric that was mapped to X. Function $worseOf(S)$ returns the worse value of a set of values.

2. P_i has the constraint $X = v_1$ and P^D has the constraint $X = v_2$. In this case, P^D's constraint is violated if and only if: $worse(v_1, v_2)$.

3. P_i has the constraints $X! = v_1, X! = v_2, \ldots, X! = v_n$ and P^D has the constraint $X = v$. In this case, P^D's constraint is violated if and only if: $worse(worseOf(domain(X) - \{v_1, v_2, \ldots, v_n\}), v)$.

4. P_i has the constraints $X! = v_1, X! = v_2, \ldots, X! = v_n$ and P^D has the constraints $X! = v'_1, X! = v'_2, \ldots, X! = v'_m$. In this case, P^D's constraints are violated if and only if: $worse(worseOf(domain(X) - \{v_1, v_2, \ldots, v_n\}), worseOf(domain(X) - \{v'_1, v'_2, \ldots, v'_m\}))$.

For the second case, we have again four cases to consider (excluding $op_1 = <$ and $op_2 = >$ for simplicity) :

1. P_i has the constraint $X \geq a$ and P^D has the constraint $X \geq b$. In this case, P^D's constraint is violated if: $a < b$ and $X \uparrow$, where the $\uparrow$ symbolizes that the value monotonicity of the metric that was mapped to X is positive. $\downarrow$ means negative.

2. P_i has the constraint $X \geq a$ and P^D has the constraint $X \leq b$. In this case, P^D's constraint is violated if: $X \downarrow$ and $a \geq b$.

3. P_i has the constraint $X \leq a$ and P^D has the constraint $X \leq b$. In this case, P^D's constraint is violated if: $X \downarrow$ and $a > b$.

4. P_i has the constraint $X \leq a$ and P^D has the constraint $X \geq b$. In this case, P^D's constraint is violated if: $X \uparrow$ and $a \leq b$.

The matchmaking process takes as input the CSPs P_i and P^D of the offers and the demand and produces four types of results. For each offer's P_i it checks if it conforms to the demand's P^D by the following way: For each variable X of P^D, we check if it also exists in P_i. If no, then variable $count_1$ is increased with the number of constraints of P^D that contain variable X while variable $count_2$ is increased with the addition of the weights of these constraints. If yes, we check which constraints of P_i containing X violate which constraints of P^D also containing X. If there are violated constraints, then variables $count_1$ and $count_2$ are updated accordingly. If P_i contains better X solutions than that of P^D's, then variable $count_3$ is set to 1. If every variable X of P^D is examined, we check if $count_1$ equals the total number of constraints of P^D. If yes, then P_i is added to *fail* matches, otherwise if $count_1! = 0$ it is added to *partial* matches along with its counters. If there are no violated constraints at all, then: if $count_3 = 1$ then P_i contains better solutions than that of P_D's and it is added to *super* matches, otherwise it is added to *exact* matches. $count_1$ counts the number of violated constraints of the demand and $count_2$ represents the total weight of these constraints. These constraints do not allow some worse solutions of the offer's P_i to be part of the demand's P^D 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 contain better solutions. *Exact* offers just conform to the demand by containing a subset of the solutions of the demand.*Partial* offers do not conform to the demand because either they do not use some metrics of the demand or they contain (worse) 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 the 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.

For the second case, the matchmaking process has provided three metrics for each *partial* offer: number of demand's violated constraints, their total weight and if offer contains better solutions that the demand. So we order the *partial* list of offers according first to total weight and then to the number of violated 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 *super* or *exact* match list according to the value of the third metric and we move to the last process in order to rank them and return them. However, the user is warned that these ordered lists represent *super* or *exact* matches only if he weakens the corresponding violated constraints 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 [4] extends the one defined in [5] 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 [4].

Complete Example. To demonstrate our QoS-based WS discovery algorithm, we supply a simple example of its application to a small set of four QoS offer CSPs P_i and one demand CSP P^D. Assume

that all CSPs have the following three definitions: $X_1 :: (0.0, 86400.0] \downarrow$, $X_2 :: (0, 100000] \uparrow$ and $X_3 :: (0.0, 1.0) \uparrow$. Based on these variable definitions, assume that each CSP has the following constraints: $P_1 : [X_1 \leq 10.0, X_2 \leq 100, X_2 \geq 50, X_3 \geq 0.9]$, $P_2 : [X_1 \leq 4.8, X_2 \leq 50, X_2 \geq 40, X_3 \geq 0.95]$, $P_3 : [X_1 \leq 16, X_2 \leq 40, X_2 \geq 30, X_3 \geq 0.98]$, $P_4 : [X_1 \leq 16, X_2 \leq 50, X_2 \geq 40, X_3 \geq 0.98]$, and $P^D : [X_1 \leq 15.0, X_2 \geq 40, X_2 \leq 60, X_3 \geq 0.99]$. Moreover, assume that the WS requester does not provide weights to the constraints of his demand and associates the following weights to the three metrics/variables: $X_1 \leftarrow 0.3$, $X_2 \leftarrow 0.3$, $X_3 \leftarrow 0.4$, while $a = 0.7$ and $b = 0.3$ [4]. In addition, assume that the following utility functions are applied to the CSOPs: $uf_{X_1} = (16 - X_1)/16$, $uf_{X_2} = (X_2 - 30)/70$, $uf_{X_3} = (X_3 - 0.9)/0.1$ [4].

We now apply the four processes of our discovery algorithm. The CSP Preprocessing process has no result at all to the CSPs. The matchmaking process produces the following four results lists: $Super = [\,]$, $Exact = [\,]$, $Partial = [(O_1, P_1, 1, 2.0, 1), (O_2, P_2, 1, 2.0, 1), (O_4, P_4, 2, 4.0, 1)]$, $Fail = [(O_3, P_3]$. 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: $Super^* = [(O_1, CSP_1), (O_2, CSP_2^O)]$ and $Partial = [O_4, CSP_4^O]$. Finally, the selection process solves two CSOPs for each of the first two offers, produces one score for each: $Score_1^O \simeq 0.44$ and $Score_2^O \simeq 0.55$, and returns the following ordered list to the requester: $Super^* = [(O_2, 0.55), (O_1, 0.44)]$. However, the WS requester is warned that this list contains offers that do not satisfy his constraint: $X_3 \geq 0.99$.

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 will have been achieved.

6.2 Generic Discovery Algorithm

This algorithm is more generic than the previous one as it allows any kind of constraint to be used in the CSPs. However, now the notion of better or worse applied to CSP solutions is altered. To remind, the previous algorithm checks if a tuple of a n-tuple solution of an offer is worse than all corresponding tuples of the solution set of the demand in order to find a mismatch/conflict. The generic discovery algorithm checks if the whole solution of the offer is worse than all solutions of the demand by assigning a preference or value to each CSP solution. So it is more closed to the definition of conformance we have previously given in this section.

One question is how the assignment of preferences to solutions takes place. The technique we use is based on utility functions and weights on CSP variables [5]. Each CSP variable (a map of a metric) is given a (user) weight or preference (taking values from the set $[0.0, 1.0]$) to reflect the significance of this variable to the preference/value of the solution. In addition, each possible value of this variable is given also a preference ($\in [0.0, 1.0]$) by the variable's utility function. The preference of a CSP solution is given by the following sum on all variables X_j: $p_s = \sum_{X_j}(w_{X_j} \cdot uf_{X_j}(v_{X_j}))$, where w_{X_j} is the weight of the variable X_j, $uf_{X_j}()$ is its utility function and v_{X_j} is its value.

Based on the above technique, a partial ordering of all solutions of a CSP can be inferred. This is the appropriate mean in order to define matchmaking: an offer's CSP P_i matches the CSP P^D of the demand if its worst solution has a preference of greater or equal value with respect to the preference of the worst solution of the demand. This definition leads to two main observations: **a)** CSOPs for offers and demands have to be solved in order to find the preference of the worst solution; **b)** constraints are only used to reduce the domain of the variables. The second observation hides an important conclusion: constraint relaxation is inherent to the optimization of CSPs based on preference functions. To explain, a matching offer may have a (worst) solution that violates constraints of the demand affecting one or more variables of less significance. However, this solution surely provides better values for variables of higher significance/preference. It is like relaxing some constraints of the demand in order to match this offer. The next paragraph provides a sketch of the QoS-based WS discovery algorithm, while the last one provides a simple example of its application.

Algorithm. [**Matchmaking**] We compute the preferences $p_{s_1}^D$ and $p_{s_2}^D$ of the demand's CSP P^D worst s_1^D and best s_2^D solution respectively by solving two CSOPs (minimization and maximization) [4]. For each offer's CSP P_i, we compute the preferences $p_{s_1}^i$ and $p_{s_2}^i$ of its worst s_1^i and best s_2^i solution respectively in the same manner as above. Then, we consider four cases:

1. If $(p_{s_2}^i \leq p_{s_1}^D)$, then the offer is put in the *fail* match list.

2. If $(p_{s_2}^i > p_{s_1}^D \wedge p_{s_1}^i < p_{s_1}^D)$, then the offer is put in the *partial* match list.

3. If $(p^i_{s_1} \geq p^D_{s_1} \wedge p^i_{s_2} \leq p^D_{s_2})$, then the offer is put in the *exact* match list.

4. If $((p^i_{s_1} \geq p^D_{s_1} \wedge p^i_{s_2} > p^D_{s_2}) \vee (p^i_{s_1} \geq p^D_{s_2}))$, then the offer is put in the *super* match list.

The first case expresses the fact that the offer's best solution is not better than the worst solution of the demand and justifies the classification of the offer as *failed*. The second case expresses the fact that the offer has some *bad* solutions but also some *good* solutions so it is considered as a *partial* result. The third case concerns offers that contain a subset of the solutions of the demand and justifies their classification as *exact*. The last case is about offers that contain not only solutions of the demand but also better ones. That's why they are classified as *super* results/matches.

[**Selection**] In this process, either the best two categories of results (if not empty) or the third category are ordered based on the weighted sum of the preferences of their worst and best solutions [4].

Example. In order to show that this algorithm is more generic than the previous proposed one, we apply it on the same example. For each offer CSP P_i we have the following preferences: $[P_1 : p^1_{s_1} = 0.1982, p^1_{s_2} = 1.0]$,$[P_2 : p^2_{s_1} = 0.4528, p^2_{s_2} = 0.7857]$, $[P_3 : p^3_{s_1} = 0.32, p^3_{s_2} = 0.7428]$, $[P_4 : p^4_{s_1} = 0.3628, p^4_{s_2} = 0.7428]$. The demand's CSP P^D has the following preferences: $P^D : [p^D_{s_1} = 0.4216, p^D_{s_2} = 0.8285]$. So the discovery algorithm will produce the following results lists: $Super = [\,]$, $Exact = [O_2]$, $Partial = [(O_1), (O_3), (O_4)]$, $Fail = [\,]$.

As it can be seen, offer O_2 is in the *Exact* match list although it violates the last constraint of the demand. The reason for this is that the preference of its worse solution is greater than the preference of the worse solution of the demand. To put it in another way, O_2 provides a far better lowest value for the X_1 attribute with respect to the worse lowest value for the X_3 attribute (compared to the corresponding lowest values of the demand). Another observation is that O_1 pays the penalty of providing the minimum possible value for the X_3 attribute and is considered a *partial* result. The last observation is that O_3 is promoted as a *partial* result. O_3's promotion is due to the fact that its worse solution violates only in a small amount the constraints of the demand.

7 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, while the Java programming language is used as a bridge between them. Pellet is chosen because it is on 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 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.

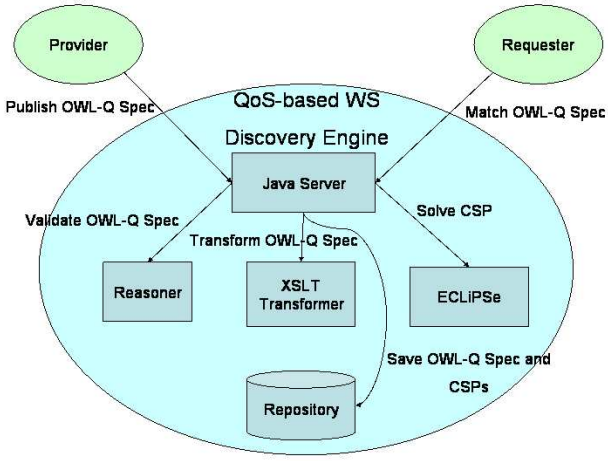

Figure 1. QoS-based WS Discovery Engine.

8 Future Work

As future work, we plan to evaluate our metric matching and discovery algorithms in order to show their performance and accuracy. We also intend to exploit advanced techniques for solving over-constrained problems like semi-ring based constraint satisfaction [1] as alternatives to the branch-and-bound algorithm used for constraint optimization solving. In addition, we plan to extend OWL-Q with the description of context of both the WS and the WS 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] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[2] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. Technical Report RC22456 (W0205-171), IBM, 2002.

[3] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

[4] K. Kritikos and D. Plexousakis. Semantic qos metric matching. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 265–274, Washington, DC, USA, 2006. IEEE Computer Society.

[5] O. Martín-Díaz, A. R. Cortés, D. Benavides, A. Durán, and M. Toro. A quality-aware approach to web services procurement. In B. Benatallah and M.-C. Shan, editors, *Technologies for E-Services (TES)*, volume 2819 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2003.

[6] E. M. Maximilien and M. P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[7] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic ws-agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2006. ACM Press.

[8] S. Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[9] M. Tian, A. Gramm, M. Nabulsi, H. Ritter, J. Schiller, and T. Voigt. Qos integration in web services. Gesellschaft fur Informatik DWS 2003, Doktorandenworkshop Technologien und Anwendungen von XML, October 2003.

[10] V. Tosic, B. Pagurek, and K. Patel. Wsol - a language for the formal specification of classes of service for web services. In L.-J. Zhang, editor, *ICWS*, pages 375–381. CSREA Press, 2003.

[11] P. Van Hentenryck and V. Saraswat. Strategic directions in constraint programming. *ACM Computing Surveys*, 28(4):701–726, 1996.

[12] C. Zhou, L.-T. Chia, and B.-S. Lee. Daml-qos ontology for web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 472, Washington, DC, USA, 2004. IEEE Computer Society.