OWL-Q for Semantic QoS-based Web Service Description and Discovery

Kyriakos Kritikos and Dimitris Plexousakis

Foundation of Research and Technology, Heraklion, Greece,
kritikos,dp@ics.forth.gr

Abstract. Semantic Web Services are emerging for their promise to produce a more accurate and precise Web Service discovery process. However, most of research approaches focus only on the functional part of semantic Web Service description. The above fact along with the proliferation of Web Services is highly probable to lead to a situation where Web Service registries will return many functionally-equivalent Web Service advertisements for each user request. This problem can be solved with the semantic description of QoS for Web Services. QoS is a set of non-functional properties encompassing performance and network-related characteristics of resources. So it can be used for distinguishing between functionally-equivalent Web Services. Current research approaches for QoS-based Web Service description are either syntactic or poor or non-extensible. To solve this problem, we have developed a rich and extensible ontological specification called OWL-Q for semantic QoS-based Web Service description. We analyze all OWL-Q parts and reason that rules should be added in order to support property inferencing and constraint enforcement. Finally, we line out our under-development semantic framework for QoS-based Web Service description and discovery.

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 [1], 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. Actually, the current modeling efforts of QoS specifications only differ in the expressiveness of these constraints. However, these efforts fail in QoS metric 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 produce irrelevant or incomplete results.

There are two main approaches for QoS-based Web Service Discovery. The first one, analyzed in [2], relies on the subsumption of the compared QoS-based WS descriptions for matchmaking. However, as indicated by the authors of this approach, subsumption is quite slow and additional techniques must be devised for speeding it up. The other approach, analyzed in [3], transforms the compared QoS-based WS descriptions to a Constraint Satisfaction Problem (CSP) [4] and then solves this problem. This approach has been shown [3] to be quick and efficient in realistic scenarios. In addition, tools for CSP solving are more mature than reasoning tools. Thus, the second approach is more appropriate for QoS-based WS discovery. Unfortunately, this approach also suffers from some shortcomings that will be analyzed in detail in the sequel of this paper.

Based on the above deficiencies, we have developed OWL-Q [5], a rich, extensible and modular ontology language that complements the WS functional description language OWL-S. In addition, we have extended the most prominent CSP-based QoS-based WS discovery approach [3]. In this paper, after reviewing the state-of-the-art in QoS-based WS description and discovery, we analyze in detail all parts of OWL-Q, as OWL-Q's design has been finalized. Next, we explain that OWL cannot be used for reasoning about relations between properties and for enforcing constraints so as to justify the extension of OWL-Q with SWRL rules. In addition, we provide examples of types of rules that have been added to OWL-Q. Then, we analyze our QoS metric matching and alignment and CSP-based WS Discovery algorithms [5, 6] and we shortly describe the building tools 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. While *OWL-S* is a standard *semantic* approach for WS Description, it does not describe any QoS concept.

Ran [7] proposes a syntactic extension to UDDI for QoS-based WS description. Maximilien and Singh [8] 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. [9] 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) [10] 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. [11] 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. [12] 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. [2] 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. [3] 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 a *Constraint Satisfaction Optimization Problem* (CSOP) [4].

3 QoS-based Web Service Description

3.1 Requirements for QoS-based Web Service Description

After reviewing related work in QoS-based WS Description, we have come up with the following requirements that must be satisfied by a QoS-based WS description language [13]:

- *Devise an extensible and formal semantic QoS model*
- *Comply with standards*
- *Support the syntactical separation of QoS-based and functional parts of service specification*
- *Support refinement of QoS specifications and their constructs.*
- *Allow both provider and requester QoS specification*
- *Allow fine-grained QoS specification*
- *Devise an extensible and formal QoS metrics model*
- *Devise a corresponding extensible and formal QoS attributes, units, functions and measurement directives model.*
- *Allow classes of service specification*
- *Enabling of tractable matchmaking algorithms*

3.2 OWL-Q

Based on the requirements of QoS-based WS Description we have set in the previous subsection, we have developed an OWL-S extension (*syntactical separation*), called OWL-Q [5], for QoS-based WS description of both requests and offers.

We have extended OWL-S ontological description for two reasons: to comply with Semantic WS description standards (*standards compliance*) and to use the OWL ontology formalism (*extensible and formal semantic QoS model*). OWL-Q is actually an upper ontology comprised of many sub-ontologies/facets, each of which can be extended independently of the others (*syntactical separation and refinement of QoS specifications*). Each facet concentrates on a particular part of our QoS WS description. In its new form, OWL-Q has eleven facets: OWL-Q (main), Measurement Directive, Time, Goal, Function, Measurement, Metric, Scale, QoSSpec, Unit and ValueType. In the sequel, a small analysis of each facet of OWL-Q will be provided while the most important changes with respect to its previous form will be indicated. The whole ontology will be available soon at: http://www.csd.uoc.gr/∼kritikos/OWL-Q.owl.

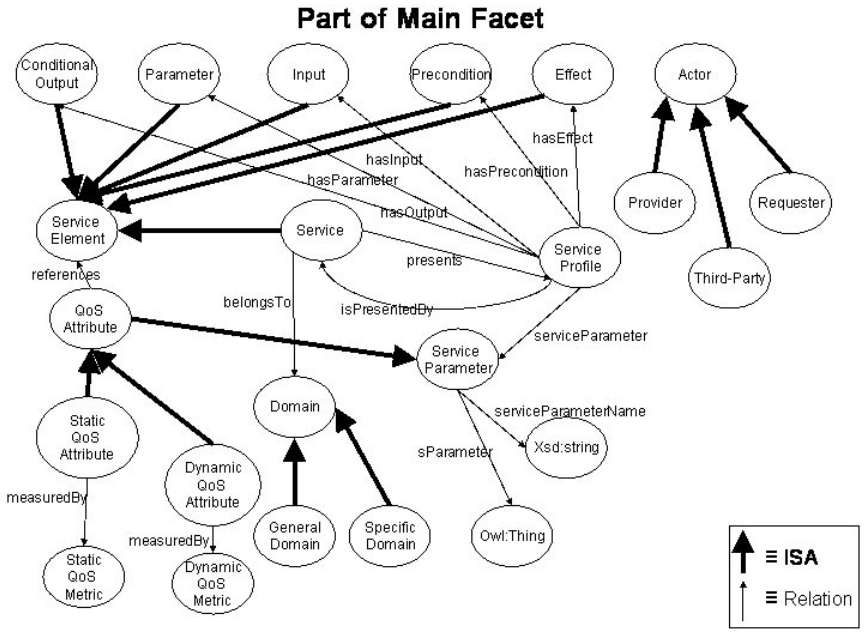

Fig. 1. Part of Main Facet.

OWL-Q (Main) Facet As can be seen in Fig. 1, the Main Facet connects OWL-S with OWL-Q and provides the high-level QoS concepts. For the connection of the two ontological descriptions, the *ServiceAttribute* class is a subclass of OWL-S *ServiceParameter* and references a *ServiceElement*. Subclasses of the latter class are *ConditionalOutput*, *Parameter*, *Input*, *Precondition*, *Effect*, and *Service*. That is a *ServiceAttribute* can reference any *ServiceElement* of a service's functional description (*fine-grained QoS specification*). Another point of

connection is that a *ServiceProfile* contains one or more *QoSOffers*'s or one QoS Request (*classes of service requirement*). A final point of Connection is that the *Actor* class is separated into three subclasses: *Provider*, *Requester*, *ThirdParty* so as to define the main actors involved in QoS-based WS description and measurement. A Service Attribute contains two subclasses: QoSAttribute and ContextAttribute and is a subclass of the general class *Attribute*. An attribute can be separated into **a)** physical or service attributes, **b)** measurable or unmeasurable attributes and **c)** unique or derived attributes. Physical attributes like *Time*, *Temperature* and *Location* characterize environmental (contextual) factors of a WS or its requester while service attributes like *Availability* or *NumOfInterfaces* are functional or non-functional characteristics of a WS. Measurable attributes like *Time* are measured by specific metrics while unmeasurable attributes like *Manageability* cannot be measured. Unique attributes like *Time* are not derived by other attributes and are measured by resource metrics while derived attributes like *Throughput* are produced by complex metrics computed by functions using metrics of other attributes. The *Domain* class represents the domain of knowledge that a service applies to and is separated into two subclasses: **a)** *GeneralDomain* and **b)** *SpecificDomain*. The *GeneralDomain* stands for every possible WS. Specific Domain can be further specialized/subsumed, for example a possible subclass could be the Travel domain. The *Value* class represents any possible integer, double, string or list-based value, it is subsumed by special symbol classes like Infinity or Limit, and it is mainly used in Goal, Value Type and Measurement definitions. Other classes that are defined but are unfolded in separate sub-ontologies are: *Function*, *Measurement*, *MeasurementDirective*, *Metric*, *QoSSpec*, *Scale*, *Schedule*, *Trigger*, *Unit*, *ValueType* and *Goal*.

QoSSpec Facet In this facet, the classes representing QoS offers and requests are defined. The class *QoSSpec* is separated into two subclasses: *QoSOffer* and *QoSDemand* in order to enable WS providers and requesters to define in the same way their QoS constraints *(both provider and requester QoS specification)*. Of course, the WS requester is enabled not only to specify constraints (by the *QoSDemand* class) but also to provide weights to metrics of his interest (by the *QoSSelection* class). The *QoSSelection* class is actually a list of <metric, weight> entries. The *QoSSpec* class represents the actual QoS description of a WS. It describes the security and transaction protocols used (URIs), the cost of using the service (double) and the associated currency for the cost (unit), the validity period of the offer or demand (*CalendarClockInterval* class of the time ontology [14]) and a list of conjunctive QoS goals/constraints with their weights (value of 2.0 if it is a hard constraint or a value in $(0.0, 1.0)$ if it is soft).

Goal Facet Mathematical formulas and QoS goals/constraints were previously expressed in OpenMath (http://www.openmath.org). Now this has changed due to change of philosophy regarding the QoS Metric Matching Algorithm described in the next section. QoS constraints are expressed in the following form: $(f(arguments)|metric)$ *op value*, where f is a function, arguments are a list of functions, metrics and values, op can be one of $\leq$, $\geq$, $<$, $>$, $=$, $!=$. For exam-

ple, the fact that metric M is less than 0.1 could be expressed by the user as: $M \leq 0.1$, where op $=\leq$. An appropriate interface will be provided to the user in order to enable him to specify constraints in our user-friendly customized expression form.

Measurement Facet Measurements are now modeled in OWL-Q 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 Measurement class in OWL-Q contains a single value (*Value* class), it is produced by an *Actor* at a specific time point (*CalendarClockDescription* class of [14]), it concerns a specific *Metric* and belongs to a specific party (*Actor*).

Function Facet Functions in OWL-Q are separated into functions applied to metrics (for producing complex metrics or checking satisfiability of their constraints) and functions applied to scales. Metric functions have specific arity and contain arguments that are either Metrics, values of other Metric Functions. Scale transformation functions are further categorized into five disjoint subclasses and are used for converting one scale expression to an expression of another scale.

Measurement Directive Facet The *MeasurementDirective* class specifies the way simple metrics are measured. It specifies by a URI how the value of a managed resource is going to be achieved and by a *ValueType* the type of the return value. In addition, it specifies if the party responsible for the measurement will ask for the value or get it when it is ready (i.e it specifies the access model, where $AccessModel = Pull \cup Push$). This class can have many subclasses, some of which may require a possible extra attribute (*timeOut*) specification concerning the time duration (*DurationDescription* of [14]) that the measurement party will wait to get the measurement value (consider for example the *Status* measurement directive [10]).

Time Facet This facet specifies the *Schedule* and *Trigger* classes. A schedule is used to describe the frequency (*frequency* has range *DurationDescription*) and time interval (*interval* has range *CalendarClockInterval* [14]) of a complex metric computation. Alternatively, a complex metric computation can be executed at a specific time point (*CalendarClockDescription* [14]), information that is encapsulated in a trigger definition.

Metric Facet The Metric Facet describes all the appropriate classes and properties used for a proper formal definition of a QoS metric (*QoS metric model*). This metric facet is actually an upper ontology representing any abstract QoS metric. A specific QoS metric can be created by refining the *QoSMetric* class. Many specific QoS metrics (especially the general ones) can be part of a midlevel ontology created for QoS metric reuse. We prefer specialization to instantiation because it allows for a quicker reasoning process. We plan to develop a mid-level

ontology defining cross-domain QoS metrics and a low-level ontology for defining QoS metrics for particular domains.

The *QoSMetric* is one of the most important classes of OWL-Q representing a QoS metric. The values of a QoS metric are provided by an *Actor*. A QoS metric belongs to a *Domain* of knowledge. It has only one name. It measures a $QoSAttribute \cup MeasurableAttribute$ on a specific *ServiceElement*. The value type of a *QoSMetric* is an instance of the *ValueType* class (analyzed in a separate facet) while the scale of the value is an instance of the *Scale* class. A *QoSMetric* is separated into static and dynamic metrics. A *StaticQoSMetric* is computed only once according to a *Trigger* in order to produce a value for a *StaticQoSAttribute*. A *DynamicQoSMetric* is computed repeatedly according to a *Schedule* to produce values of a *DynamicQoSAttribute* that change over time. It can be a simple QoS metric *measuredBy* a *MeasurementDirective* or a complex one. *ComplexMetric*s are derived from other metrics with the help of a *MetricFunction*. Last but not least a *QoSMetric* is related to other metrics according to two types of *Relationship*s: *Independent* and *Related*. When two metrics are related, we can specify the direction of their values or the impact of one's value to the other's value. According to the scale it uses, a metric can be categorized into absolute, interval, nominal, ordinal and ratio metrics. Ratio metrics directly reference a Unit of measurement as a *RatioScale* is actually equivalent to a *Unit*. Metrics can be positively or negatively monotonic. In this way, we know if one metric value is better than another one.

Scale Facet A measurement scale controls the value type and the type of operations allowed for a metric and belongs to a specific *Attribute*. It also specifies the way one value expression bound to one scale can be transformed to another value expression of another compatible scale (both scales belonging to the same metric). So specific scales can be compatible if they belong to the same scale type and there is a *ScaleTransformationFunction* that transforms their expressions into each other. *Scale* is a more general notion with respect to *Unit*. A scale can be categorized into five disjoint subclasses: *NominalScale*, *OrdinalScale*, *IntervalScale*, *RatioScale* and *AbsoluteScale* [15]. Nominal scales concern metrics that have as value type a set of numbers or strings. The members of this set cannot be compared (no ordering). Specific nominal scales can be compatible if there is a one-to-one mapping function between their corresponding value types. Ordinal scales apply to metrics that have an ordered set as value type. Metrics belonging to different ordinal scales cannot be added, multiplied, divided or abstracted in QoS constraints. We can transform one ordinal scale expression into another one with the help of monotonic functions. Interval scales preserve not only ordering but also differences. However, they do not preserve ratios. The operations of addition and substraction are allowed between different ordinal metrics. We can transform one interval scale expression into another one with the help of affine transformation functions of the form: $M = a * M^{'} + b$. Ratio scale preserve ordering, size of intervals and ratios. In a ratio scale there is always a zero element representing the total lack of the measured attribute. All arithmetics are allowed between different ratio metrics. We can transform

one ratio scale expression into another one with the help of mapping functions of the form: $M = a * M^{'}$. Finally, the following facts are true for an absolute scale: **a)** measurement is made simply by counting the number of elements in the measurement set; **b)** measured attribute takes the form: "num of occurrences of x in the entity"; **c)** all arithmetic analysis is meaningful; d) the set of acceptable transformations between different absolute scale expressions is the identity transformation function.

Unit Facet The Unit Facet formally describes the unit of a ratio scale of a ratio QoS metric. A *Unit* has one name, several abbreviations and synonyms (even in different languages). A *Unit* belongs to a *System of Units*, which system can be *SelfConsistent* or *NonSelfConsistent*, and is associated with the same *QoSAttribute* as the one that is measured by the QoS metric of the unit. A *Unit* is separated into *BasicUnit*s and *MultipleUnit*s. The *BasicUnit* class is separated into *UniqueAttributeUnit*s and *DerivedAttributeUnit*s, depending on the type of *Attribute* measured. A *MultipleUnit* is associated with a *BaseUnit* and converted to it by a constant (*magnitude*). It has a name composed of the name of its *BaseUnit* and a prefix. A *DerivedUnit* is proportional to some *Unit*s and inverse proportional to other *Unit*s. It also has a *magnitude* that is used to express its mathematical definition in relation to the other (inverse) proportional units. An unit is equivalent to another unit and can be converted to it with the help of their ratio scale and its ratio transformation functions.

Value Type Facet The *ValueType* ontology describes the types of values a QoS metric can take. The *ValueType*s can be *Scalar* or *NumericUnion*, or *ListBased* types. *Scalar* value types are simple value types that can be *Numeric* or *String*. *ConstrainedNumeric* value types represent *Numeric* value types that have (upper, low or one) limits (e.g. the Integers set [2,5] or the Integer value {2}). The *NumericUnion* class represents value types that are expressed as unions of *Numeric* value types (e.g. $[1, 2] \cup \{4\} \cup [9, 11]$). The *List-Based* class represents list value types that have a specific size and whose elements are of a specific *ValueType*. Subclasses of the *ListBased* class are: numeric or string lists, queues and timeseries.

3.3 Rules

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 or more property or class instance facts (or a mixture of them) 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 [14] 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 and/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 (see next section) have to be specified. So we are currently in the process of extending OWL-Q with rules, which are expressed in SWRL – the most widely used SW rules proposal at present. However, most reasoners only partially support SWRL and this is a major obstacle to our under-implementation semantic framework for QoS-based WS description and discovery.

4 QoS-based Web Service Discovery Framework

4.1 QoS Metric Matching Algorithm

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 [5] 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, which is a problem area of symbolic computation, is undecidable. For this reason, we decided to use CSP solving that is decidable although computationally expensive. The trick for this transformation/change is the simple observation that symbolic expression equality can be seen alternatively as unsatisfiability of a CSP containing a constraint enforcing that the difference of the two expressions is not zero. In other words, if the CSP does not have any solution, then the constraint cannot be enforced and the negation of its formula is always true. The latter infers the equality of the expressions compared, which is our goal. For example, suppose that we want to check if two expressions $(x+1)^2$ and x^2+2x+1 are equal. We can easily transform the previous problem to a CSP: $[(X : -\infty \ldots +\infty), ((x+1)^2 - x^2 - 2x - 1! = 0)]$ and try to solve it. There is no solution to this CSP, so the constraint is unsatisfiable, the difference of the two expressions is always zero and thus these expressions are equal.

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

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

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

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

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

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

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 [6].

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)) and the second (rule (4)) of the three body rules have not been altered and we are not going to further describe them.

The last of the three body rules, rule (5), compares and possibly aligns one by one the metrics from which M_1 is derived with the corresponding metrics of M_2 and updates appropriately the measurement formulas of M_1 and M_2. Then from the derivation lists of M_1 and M_2 and their measurement formulas a (possibly non-linear) CSP is defined and solved. More details about the algorithm can be found in [6].

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: $smatch(M_1, M_2)$, $rcm(Downtime_1, Downtime_2)$, $rcm(Uptime_1, Uptime_2)$. 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. The fourth clause infers that: $rcm(Downtime_1, Downtime_2)$, $rcm(Uptime_1, Uptime_2)$. So $Downtime_1$ and $Downtime_2$ are mapped to a new metric $Downtime$ and $Uptime_1$ and $Uptime_2$ are mapped to $Uptime$. In this way, it stands that: $M_1.derivedFrom = M_2.derivedFrom = [Downtime, Uptime]$, $M_1.measuredBy = 1 - Downtime/(Downtime + Uptime)$, $M_2.measuredBy = Uptime/(Uptime + Downtime)$. The last clause of the rule will create and solve a CSP that has the following definitions: $Downtime, Uptime :: [0.0, 1.0]$ and constraints: $1 - Downtime/(Downtime + Uptime) - Uptime/(Uptime + Downtime)! = 0$. This CSP is unsatisfiable so finally the fact $match\,(M_1, M_2)$ is inferred.

4.2 QoS Metric Alignment Algorithm

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. More details about this algorithm and how the transformation of a QoS spec S to a CSP is carried out can be found in [6].

4.3 QoS-based Web Service Discovery Algorithm

One of the most prominent QoS-based WS discovery algorithm [3] 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 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; **4)** it does not return any result when QoS requests are over-constrained, where over-constrained problem specifications happen very often in the real-world.

Matchmaking of QoS offers and demands is based on the concept of conformance [3], which is mathematically expressed by the following equivalency:

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

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 $(0.0, 1.0)\uparrow$, where $\uparrow$ denotes that this type is positively monotonic i.e. greater values are better than lower ones. 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 [3] and the new definition of matchmaking, we have proposed two QoS-based WS discovery algorithms [6]. 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. Due to space limitations of this paper, we are going to analyze only the second algorithm.

Generic Discovery Algorithm This 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.

The big 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 [3]. 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 $p^D_{s_1}$ and $p^D_{s_2}$ of the demand's CSP P^D worst s^D_1 and best s^D_2 solution respectively by solving two CSOPs (minimization and maximization) [5]. For each offer's CSP P_i, we compute the preferences $p^i_{s_1}$ and $p^i_{s_2}$ of its worst s^i_1 and best s^i_2 solution respectively in the same manner as above. Then, we consider four cases:

1. If $(p^i_{s_2} \leq p^D_{s_1})$, then the offer is put in the *fail* match list.
2. If $(p^i_{s_2} > p^D_{s_1} \wedge p^i_{s_1} < p^D_{s_1})$, 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 [5].

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 $b = 0.3$ [5].

In addition, assume that the following utility functions are applied to the CSOPs: $uf_{X_1} = (16 - X_1)/16$, $uf_{X_2} = (X_2 - 30)/70$, $uf_{X_3} = (X_3 - 0.9)/0.1$ [5].

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. 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.

4.4 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 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 supports the tasks of ontology validation and reasoning, OWL 1.1 datatype reasoning and partial SWRL inferencing. ECLiPSe is chosen as it supports advanced linear constraint solving and extends the common facilities of Prolog. Additionally, it can be extended to support non-linear constraint solving through external solvers. More details about the architecture and the functionality of the main components of the discovery engine can be found in [6].

5 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 [16]. We also plan to extend OWL-Q with the description of the context of both the WS and the WS requester so as to achieve Context-aware QoS-based WS discovery. Our ultimate and final goal is to accomplish 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 '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 915–922

2. Zhou, C., Chia, L.T., Lee, B.S.: Daml-qos ontology for web services. In: ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04), Washington, DC, USA, IEEE Computer Society (2004) 472
3. 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) (2005) 439–468
4. Van Hentenryck, P., Saraswat, V.: Strategic directions in constraint programming. ACM Computing Surveys **28**(4) (1996) 701–726
5. Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: ECOWS '06: Proceedings of the European Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2006) 265–274
6. Kritikos, K., Plexousakis, D.: Semantic qos-based web service discovery algorithms. In: ECOWS '07: Proceedings of the European Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2007) (accepted).
7. Ran, S.: A model for web services discovery with qos. SIGecom Exch. **4**(1) (2003) 1–10
8. Maximilien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Rec. **31**(4) (2002) 36–41
9. 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, CSREA Press (2003) 375–381
10. Keller, A., Ludwig, H.: The wsla framework: Specifying and monitoring service level agreements for web services. Technical Report RC22456 (W0205-171), IBM (2002)
11. 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)
12. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic ws-agreement partner selection. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, ACM Press (2006) 697–706
13. Kritikos, K., Plexousakis, D.: Requirements for qos-based web service description and discovery. compsac **2** (2007) 467–472
14. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Trans. Asian Lang. Inf. Process. **3**(1) (2004) 66–85
15. Fenton, N.E.: Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, Boston, MA, USA (1996)
16. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM **44**(2) (1997) 201–236