Quality of service requirements specification using an ontology

Quality of Service Requirements Specification Using an Ontology

Glen Dobson
Computing Department, Lancaster University, Lancaster, UK
g.dobson@lancs.ac.uk

Russell Lock
Computing Department, Lancaster University, Lancaster, UK
r.lock@lancs.ac.uk

Ian Sommerville
Computing Department, Lancaster University, Lancaster, UK
is@comp.lancs.ac.uk

Abstract

This paper describes an approach to specifying the QoS requirements of service-centric systems using an ontology for Quality of Service. A requirements matching tool which makes use of this approach is also presented. This tool allows clients to discover services and differentiate between them based upon their QoS requirements. The ways in which the tool both constrains and aids the user in constructing requirements, and applies a degree of intelligence in the matching process, demonstrates the advantages of the underlying ontology.

1. Introduction

A service-centric system may have a static architecture decided at design-time. Service-Oriented Architectures (SOA) also open up the possibility of a dynamic architecture where certain components are only discovered and bound at runtime.

Both of these approaches ideally require a service marketplace, in which functionally equivalent services compete for the same custom. In such a situation non-functional characteristics, i.e. Quality of Service (QoS), become more important in distinguishing between the competing services.

In either case, it is also desirable to have a machine understandable QoS vocabulary. In the static case this would prove useful in implementing design-time tools, in the dynamic case it is absolutely necessary in order to perform service selection at runtime based upon service QoS. A shared QoS vocabulary also aids in representing SLAs, provider QoS advertisements and client-side QoS requirements in a consistent and interoperable manner. Below is a list of stakeholders, designed to illustrate the requirements of different parties for a shared QoS vocabulary:

Requirements common to both service and client:

- To provide common reference for negotiation between provider and client
- To develop legally binding agreements on service utilisation
- To determine unambiguous monitoring responsibilities
- To provide unequivocal legal backing should a agreement be broken

Service Provider

- To advertise services in the electronic marketplace
- To aid in internal service composition and aggregation of sub services

Client

- To differentiate between services both at design-time and runtime

This paper describes an ontology which provides the basis for such a shared vocabulary, whilst also enabling a degree of machine “understanding” of the concepts represented. The paper is structured as follows: Section 2 explains the concept of an ontology and some related technologies; Section 3 details the structure of QoSOnt itself; Section 4 explains the limitations of OWL for representing QoS requirements; Section 5 describes a requirements matching tool which makes use of QoSOnt; Section 6 compares QoSOnt to related work in the field; Section 7 evaluates QoSOnt; Section 8 suggests future work; and finally Section 9 draws conclusions on the work.

2. Background

2.1. Ontologies in Software Engineering

In software engineering, an ontology can be defined as "*a specification of a* conceptualization" [1]. More precisely, an ontology is an explicit formal specification of how to represent the objects, concepts, and other entities that exist in some area of interest and the relationships that hold among them. In general, in order to be useful, an ontology must represent a shared, agreed upon conceptualisation.

The use of ontologies in computing has gained popularity in recent years for two main reasons:

1. They facilitate interoperability.
2. They facilitate machine reasoning.

In its simplest form an ontology is simply a taxonomy of domain terms. However, taxonomies by themselves are of little use in machine reasoning. The term ontology also implies the modeling of domain rules. It is these which provide an extra level of machine "understanding".

Ontologies are already used to aid research in a number of fields. One example is the National Cancer Institute Thesaurus [2], which contains over 500,000 nodes covering information ranging from disease diagnosis to the drugs, techniques and treatments used in cancer research. Ontologies are also often used in the development of thesauri which need to model the relationships between nodes.

The constructs used to create ontologies vary between ontology languages. One class of ontology languages is those which are based upon description logics [3]. OWL is one such language. This language is discussed in the following section as a concrete example of how an ontology may be created.

2.2. OWL

OWL [4] is the Web Ontology Language - an XML-based language for publishing and sharing ontologies via the web. OWL originated from DAML+OIL both of which are based on RDF (Resource Description Framework) triples. There are three 'species' of OWL – but the most useful for reasoning - OWL-DL - corresponds to a description logic.

An OWL ontology consists of Classes and their Properties. The Class definition specifies the conditions for individuals to be members of a Class. A Class can therefore be viewed as a set. The set membership conditions are usually expressed as restrictions on the Properties of a Class. For instance the allValuesFrom and someValuesFrom property restrictions commonly occur in Class definitions. These correspond to the universal quantifier (∀) and existential qualifier (∃) of predicate logic. More precisely, in OWL such restrictions form anonymous Classes of all individuals matching the corresponding predicate.

Classes may be constructed from other Classes using the intersectionOf, unionOf and complementOf constructs which correspond to their namesakes from set theory. Another way to define a Class is to specify all individuals of which it consists explicitly using the oneOf construct.

A key feature of OWL and other description logics is that classification (and subsumption relationships) can be automatically computed by a reasoner. An open world assumption is made. This means that no assumptions are made about anything which is not asserted explicitly. One outcome of this is that a Class definition does not act as a template for individuals as it might in a closed world. For instance, an individual may have extra Properties about which nothing is asserted in its Class definition. An individual may also be a member of many Classes.

Because classifications can be inferred, the creator of an individual does not need to be aware of all possible Classes into which the individual may fall at the time of creation. Instead, all Classes of which it is a member can be inferred by a reasoner.

The following snippet from our ontology gives a flavour of OWL. It defines a Class MeasurableAttribute, stating that it is exactly equivalent to the QoSAttribute Class intersected with the set of all individuals which have a Property "hasMetric", with at least one value which is a "Metric"; intersected with the set of all individuals which have a property "hasMetric" with only values which are "Metrics". Finally it states that the class MeasurableAttribute and UnmeasurableAttribute are disjoint.

```
<owl:Class rdf:about="#MeasurableAttribute">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="#Metric" />
          <owl:onProperty>
            <owl:InverseFunctionalProperty rdf:ID="hasMetric" />
          </owl:onProperty>
        </owl:Restriction>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Metric" />
            <owl:onProperty>
      <owl:InverseFunctionalProperty rdf:about="#hasMetric" />
            </owl:onProperty>
          </owl:Restriction>
        <owl:Class rdf:about="#QoSAttribute" />
      </owl:intersectionOf>
    </owl:Class>
```

```
  </owl:equivalentClass>
  <owl:disjointWith>
<owl:Class rdf:ID="UnmeasurableAttribute" />
  </owl:disjointWith>
</owl:Class>
```

Clearly this is not particularly human-readable, especially because the Classes and Properties referenced (Metric, hasMetric, Unmeasurable-Attribute) could be defined anywhere in the file. Editing OWL manually can be difficult for the same reason. We used Protégé [5] and its OWL plug-in in our ontology development.

2.3. OWL-S

OWL was not developed to describe web services; in order to apply OWL under these circumstances OWL-S was developed. OWL-S is designed to provide many of the common markup language constructs necessary to describe services. Along with OWL and RDF it is a core "semantic web" technology. The semantic web is a movement to make the semantics of web-content accessible to machines. It has been summarised by its originators as *"an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation"* [6].

The OWL-S ontology is structured around the class Service, which consists of one or more "profiles", "groundings" and a single "model". The profile describes what a service requires and provides. The model is a functional model (i.e. it describe how the service works), whilst the grounding describes how to actually use a service (most commonly linking between the OWL-S and WSDL description).

The "profile" is the class of relevance to QoS. It is here that a service's non-functional parameters can be defined. QoSOnt is best used as an extension to OWL-S by the service provider, since OWL-S provides the ability to describe the non-QoS aspects of services. This also unifies the service specification so that it is accessible through a single point.

3. Structure of QoSOnt

The QoSOnt ontology is modular in nature in that it is structured as a set of interconnected smaller ontologies. Figure 1 summarises some of these ontologies and some of the Classes they contain. This is just a small subset of QoSOnt to give an impression of how it is used. We concentrate mainly on how to use the Class Metric as this is most relevant to requirements engineering. We use the term attribute to refer to a general QoS property (e.g. dependability, reliability, performance) and metric to refer to a specific way of measuring an attribute.

Metric, Attribute and other basic QoS concepts are defined in the base ontology. Concepts specific to some attribute can then be built into separate ontologies on top of the base concepts.

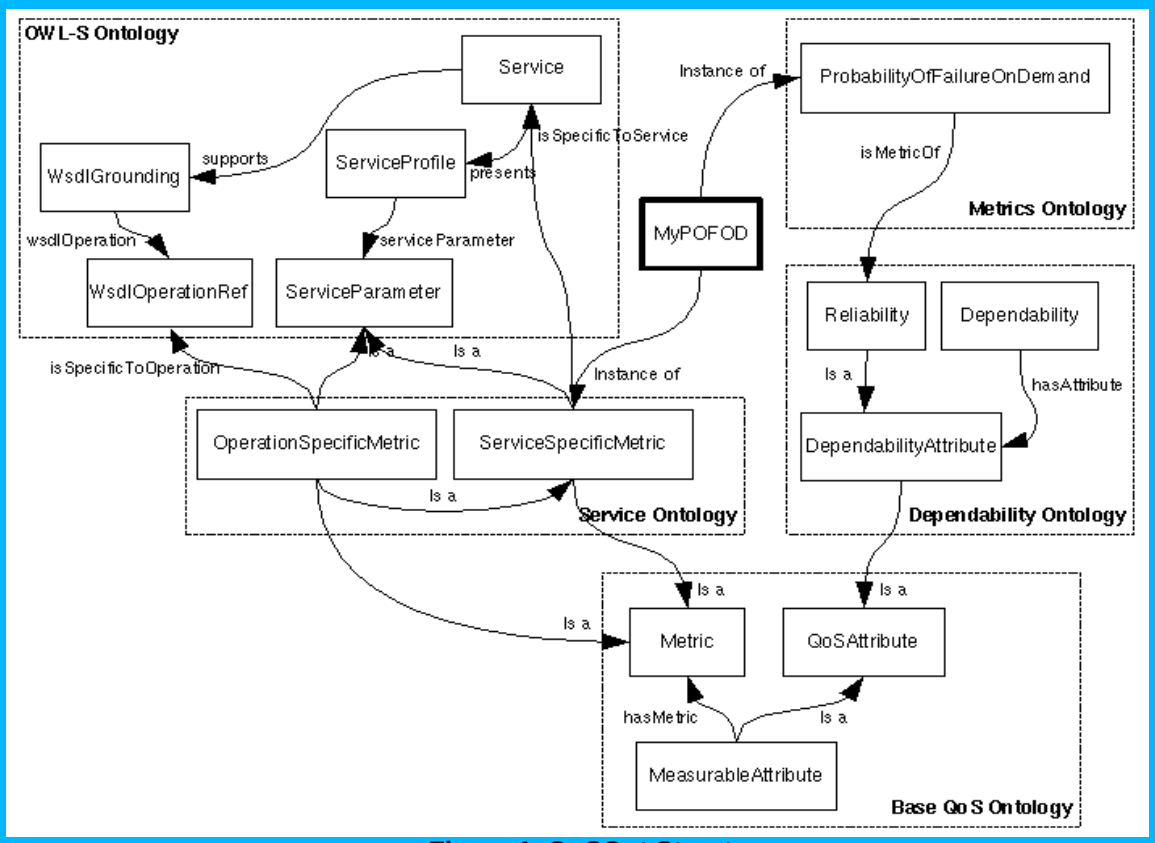

Figure 1. QoSOnt Structure

We currently have a relatively complete ontology of dependability concepts based upon [7]. We choose not to define specific metrics here as we do not wish to tie the generic concepts of dependability to specific ways of measuring dependability. We therefore have a separate ontology of actual metrics. We also have a less complete performance ontology, which plays an analogous role to the dependability ontology shown in Figure 1.

One reason for creating an ontology for a particular attribute is to allow the concepts that metrics of that attribute refer to to be defined. For instance, the concept "failure" comes in useful in order to define probability of failure on demand (POFOD) and allows POFOD to be defined for specific types of failure rather than just referring to all failures.

A further lightweight ontology ties the generic concepts of the base ontology to web services in particular. This is done by defining relationships between QoSOnt and OWL-S. This basically allows Metrics and Attributes to refer to the OWL-S Service

3.1. Metrics in QoSOnt

A Metric represents one way of measuring a specific QoSAttribute. It must result in a numerical value and must be calculable in practice as well as theory. For instance, a statement that a service has transactional throughput of 1000 transactions per second can be falsified by a single party (be they a client, provider or monitoring service) but cannot generally be measured by a client or third party, as they have no access to the traffic statistics for the service. The Class MeasurableAttribute shown in Figure 1 is a QoSAttribute having one or more associated Metrics. This is mainly a convenience Class, as the user of QoSOnt could instead subclass QoSAttribute and have a reasoner infer the more specific subclass.

A Metric is defined to consist of a description, an acceptability direction and zero or more values. The acceptability direction indicates whether higher or lower values are preferable for the Metric (e.g. A low probability of failure on demand is more desirable). It must be remembered that these Classes can be extended or constrained by their subclasses, so being over-specific at this base level is undesirable.

An individual metric conforming to a Class in QoSOnt should be created when some measured QoS data is being provided. An SLA would, in theory, also refer to such an individual metric, although we have not considered the full ramifications of using QoSOnt in SLAs. A requirement or QoS offering/advertisement should instead use a Metric Class. This is because these documents actually refer to a set of individuals (those required, or those notionally available to any client respectively) rather than any specific one. The Class in question may be directly provided by QoSOnt or created by combining, extending or restricting those Classes defined in QoSOnt.

The actual measured values of an individual Metric should be provided by some third party in order to be trusted by the client, possibly as the result of service monitoring. The provider may still maintain ownership of the rest of the document defining the individual/s - but will use OWL's import mechanism to allow the metric values to be specified externally.

Figure 1 shows an individual - MyPOFOD - and how it can be classified as both ServiceSpecificMetric and ProbabilityofFailureOnDemand. In the figure the term "instance of" is used - but the use of this term can be slightly confusing given its connotations in closed world reasoning. There is no reason that MyPOFOD (or any other individual Metric) could not simply be asserted to be equivalent to the intersection of these two Classes. However, even if this is not done, a reasoner could infer these classifications from the Properties of the individual.

A further note on the creation of an individual service Metric is that if the provider does not wish to provide an OWL-S service description then they should simply not make use of the Service and OWL-S ontologies. It is therefore generally preferable to define new Metric Classes without reference to the Service ontology, and leave it to the creator of individuals of that Metric Class to specify the intersection with the relevant Class from the service ontology. Note that in the case shown in Figure 1, MyPOFOD is classified as a ServiceSpecificMetric. It therefore refers to POFOD over the use of the whole service. Metrics specifying POFOD for specific operations could be created by intersecting with OperationSpecificMetric instead. Since OperationSpecificMetric references WsdlOperationRef, QoSont currently only supports a WsdlGrounding in OWL-S. Whilst other types of grounding are theoretically possible it is felt that the WsdlGrounding is likely to be the only one to gain widespread use.

3.2. Other QoSOnt Features

Figure 1 hides a lot of information. Among this is the fact that Metric values refer not just to some numerical datatype - but to a quantity of something in the world. In the base ontology this is modelled as a MetricValue Class with a hasUnits Property (which has the Class Unit as its range). The place this occurs most often is in defining performance Metrics. Here, it is often necessary to refer to quantities of time. For instance it is meaningless to refer to a mean time to complete of 500 without stating whether that is

measured in milliseconds or hours. We provide a time ontology for this purpose.

We also offer the possibility of unit conversion through the Class ConversionRate. This proves very useful in the situation where client requirements are stated in different units to metric data.

It is also worth noting that the interrelationships between Classes through their Properties allows traversal of the ontology to retrieve useful information. For instance, having a specific Metric, one could use its isMetricOf property to find what QoSAttribute it measures and then use the QoSAttribute's hasMetric properties to find all other metrics provided for that attribute. In a similar way one could use Class Properties to navigate to information not directly regarded as QoS (e.g. Dependability.hasMeans indicates the mechanisms used to achieve dependability in a service).

4. OWL's Limitations

Considering our preferred way of representing and matching requirements using QoSOnt reveals some shortcomings in the current OWL specification.

Recall from Section 2.2 that an OWL Class definition specifies a set of individuals of the ontology by expressing restrictions on their Properties; and that classification can be performed on individuals without the creator of the individual having any knowledge of the Class in question.

This suggests that a good way to express a QoS requirement would be as a Class created by the client. Such a requirement Class would define a subset of QoS metrics from the set of QoSOnt descriptions under consideration based upon the properties of the individual metrics (e.g. their hasValue Property).

For instance the probability of failure on demand (POFOD) requirement "POFOD<0.01" is matched by the Class defined as the intersection of POFOD and those things with a hasValue property which satisfies allValuesFrom 0.0 to 0.01. The range 0 to 0.01 could be defined using a custom XML datatype like so:

```
<xsd:simpleType name="dataRange">
  <xsd:restriction base="float">
    <xsd:minInclusive value="0.0">
    <xsd:maxInclusive value="0.01">
  </xsd:restriction>
</xsd:simpleType>
```

The OWL specification states that allValuesFrom supports quantification over a data range - but only mandates limited tool support for XML datatypes. On top of this, there is the added difficulty that there is no standard way to refer to user defined datatypes (like dataRange above) in OWL [10]. Due to these factors (among others) reasoners do not support the kind of restriction we wish to use.

This also gives us problems in modelling probabilities (values from 0.0 - 1.0) and percentages (values from 0.0-100.0), for instance, and appears to be a fundamental problem with OWL.

We envision that this problem will, in the future, be addressed in OWL and OWL reasoners. In the meantime we use a custom XML language to represent requirements. The concepts of this language map easily to those of the approach set out above in anticipation of future improvements to OWL.

5. SQRM: a QoSOnt Application

To demonstrate the use of the ontology, and aid in its evaluation, a prototype tool for service discovery and selection based upon QoS requirements has been developed. We have named the tool the Service QoS Requirements Matcher (SQRM). SQRM is designed to showcase a range of different situations in which QoSOnt can be utilised within the service domain. The tool supports the following:

- Service Discovery
- Requirement Specification
- Service Querying – Differentiation/Selection

The tool may be used at design-time to specify initial QoS requirements (and find matching services), or to narrow down a set of services already selected using some other method. The code could also form the basis for an API to allow QoS-based service selection to be performed dynamically at runtime.

5.1. Service Discovery

The first stage SQRM is designed to undertake is the initial discovery of services. UDDI [8] is used for this purpose. We do not attempt to address the existing problems with public UDDI registries (un-vetted/incorrect information etc), but instead use UDDI purely as one way to identify services worth further investigation. We implement this functionality using JAXR [9] which is registry independent - so theoretically would find it easy to implement support for other discovery mechanisms.

Clients query the repository via keyword search (see Figure 2). The extent to which clients can search for particular service requirements at this stage is highly restricted, this instead occurring at the next stage.

For each service the user selects the tool retrieves a QoSOnt document linked to the service entry

(provided this information has been published in the registry or is available through the WSDL).

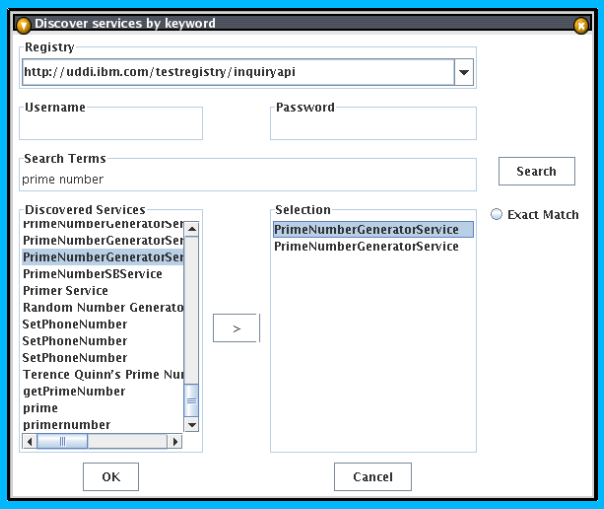

Figure 2. Service Discovery in SQRM

5.2. Requirement Specification

QoS Requirement and capability specification affects all clients and services. Without a way to specify requirements a client could not differentiate between services; without capability specification a service could not advertise its resources. The SQRM tool currently concentrates on the client viewpoint – providing a graphical means of specifying QoS requirements. Much of it however, could be reused for a provider-side specification and publishing tool.

To demonstrate the form QoS requirements may take, we briefly introduce one of the scenarios used to evaluate QoSOnt. The example used is based upon the field of epidemiology, and the study of pandemics. The computation of the projected spread of diseases on given population models is both time consuming and of interest to multiple bodies, governmental, academic and independent.

Requirements relating to different algorithms / processing capacity / time expended, make QoS specification an important factor. For example some algorithms work better with larger datasets; others may converge on an answer in such a way as to make long processing runs unnecessary for the accuracy required; for others, short runs may render results useless. Information of this type can be built into an ontology, creating a richer information resource than a mere list of supported functions.

In SQRM, a QoS requirement is basically a predicate (represented in XML), the truth value of which depends upon the asserted facts in the QoS descriptions of the client selected services. The subjects of the predicates are instances or Classes defined in QoSOnt. In contrast to requirements, the provider's description of their QoS capabilities consists of asserted propositions. These often simply say "QoS Metric X has been measured to have value Y".

Requirement predicates are visualised as a tree – the leaves of which are Values or Classes of Metric expressed in QoSOnt. The inner nodes are logical and arithmetic operators. Such as AND / OR. These allow expressions of the type shown below to be inputted:

Mean time to Failure > 10 days
AND
Mean Availability > 98%

Figure 3 shows a screenshot of the current implementation of our SQRM requirement specification environment.

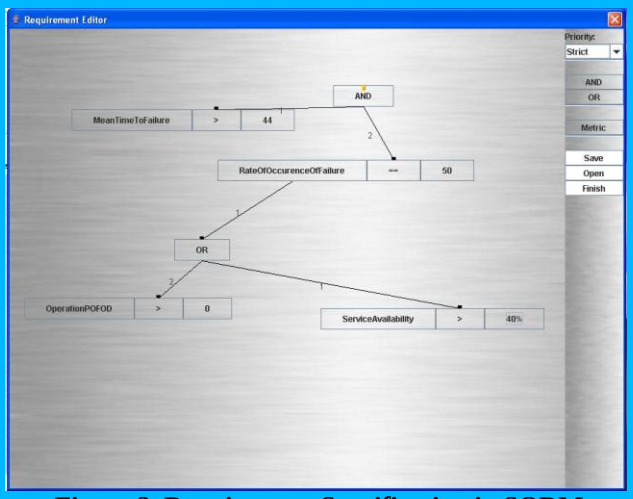

Figure 3. Requirement Specification in SQRM

5.3. Requirements Matching

Determining whether a service supports certain metrics is of limited use without being able to compare the clients' requirements against the services capabilities. The ontology becomes useful (for example) in situations where metrics are not defined in the same units between client and provider; this allows a tool to take into account and convert types with the aid of the ontology. The requirements matching phase takes files from both client and provider(s), analyses, and provides feedback to the client on the compatibility of the requirements and capabilities documents. The matching parser is designed to enable complex expressions of AND / OR construction to be used. It is not the job of the matching tool to negotiate a settlement between client and service provider, this instead would be provided by an additional phase of the service operation cycle. The requirements matcher is therefore useful as a first attempt to refine the

services available, prior to the commencement of negotiation proper.

6. Related Work

DAML-QoS [11] like our own QoSOnt, is an ontology for Quality of Service (QoS) in service-centric systems. Like QoSOnt it is realised using OWL (Web Ontology Language) (or at least its predecessor DAML+OIL) and works in symbiosis with OWL-S.

As well as QoS description, DAML-QoS supports concepts such as QoS adverts and inquiries. An attempt is made to do this in much the same way described in Section 3.2.

The approach presented in [11] also appears to be fundamentally flawed in that it uses cardinality constraints to express bounds upon QoS properties. As the term cardinality suggests, this is actually a misuse of this OWL construct. A cardinality constraint puts constraints on the number of values a property can take, not on the values themselves. Even if the approach taken was valid, it also carries the limitation that it can only express bounds as positive integers (e.g. there is no simple way to say "availability> 0.999").

The domain features modelled also seem to be rather sparse. Essentially DAML-QoS seems to be little more than a schema for QoS. As such, nothing distinguishes it from the many existing QoS specification languages [12].

[13] also presents an ontology with many similarities to our own. A framework using the ontology to support dynamic web services selection is also outlined. Despite its promise, this ontology lacks both an openly available implementation and links to OWL-S.

Our work seeks to address the gaps left by this work by providing an openly available, extensible OWL ontology, allowing complex and varying QoS metrics to be defined. Our aim is not just to provide a schema for QoS in web services - but use the power of knowledge representation in OWL to allow a certain degree of intelligence to be applied by agents and applications (e.g. conversion of units, inference of composite metric values, inference of the QoS of composite services).

7. Evaluation

The evaluation of an ontology such as QoSOnt ultimately relies upon its application by the research community. We are hoping to soon benefit from interaction with a number of other interested parties.

We see QoSOnt as something which may, in the future, form the basis of a standard QoS ontology for use across the community. During development, we have simulated its usage by generating a set of scenarios, one of which was introduced briefly in the previous section, relating to pandemics. QoSOnt aims to provide a common QoS conceptualisation for use by client, provider, and third party intermediary systems. We have therefore attempted to consider the scenarios from each of these viewpoints, although we have initially concentrated on the client and provider point of view. We consider the scalability of the ontological approach to be sufficient for the needs of large scale projects; however we do accept that the current tooling may require further work to maintain usability when scaled to deal with large numbers of requirements.

8. Future Work

In the future we hope to continue our efforts in the expansion of QoSOnt in parallel with our work on SQRM. An avenue we have begun to explore is expressing, on top of QoSOnt, how metrics aggregate under various forms of composition. We also plan to explore the way in which QoSOnt could be further leveraged in more complex QoS specification scenarios. In particular we wish to address certain limitations of common dependability metrics. The issue of moving beyond UDDI to find the best way to publish and make QoS specifications easily discoverable and queryable is also on our agenda, as is addressing the outstanding area of QoS monitoring.

Currently, the application of unit conversion is slightly cumbersome from the client's point of view. The client has to find the appropriate ConversionRate and compute the converted values themselves. We hope to use a rules language such as the Semantic Web Rules Language (SWRL) [14] to represent this knowledge in the ontology in future. SWRL would allow us to specify conversion rates as an implication between an antecendent (the value with its original units) and a consequent (the converted value). This would be transparent to the client.

Further, since we began work on QoSOnt the OWL-Time [15] ontology has matured significantly. It would therefore be useful to align QoSOnt with this ontology rather than our own time ontology.

9. Conclusion

This paper has put forward an approach to requirement specification based upon a shared QoS ontology. In order to ground the concept further, we have developed tools to leverage the benefits of an ontology for QoS, and evaluated our results against scenarios designed to test the capabilities of the design. We accept that real world examples may pose us with

unexpected situations. We are therefore seeking to collaborate with real world service users in order to further evaluate and improve QoSOnt.

10. References

[1] T. R. Gruber, “A translation approach to portable ontologies”, *Knowledge Acquisition*, Vol.5, No. 2, pp199-220, http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html, 1993

[2] National Cancer Institute (NCI) Thesaurus, http://www.mindswap.org/2003/CancerOntology/

[3] Franz Baader, Ian Horrocks, Ulrike Sattler. “Description logics as ontology languages for the semantic web”, in *Lecture Notes in Artificial Intelligence*. Springer, 2003. http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/BaHS03.pdf/

[4] W3C, "Web Ontology Language (OWL)", *http://www.w3.org/2004/OWL*

[5] DAML, “OWL-S”, http://www.daml.org/services/owl-s/

[6] Tim Berners-Lee, James Hendler, Ora Lassila, “The Semantic Web”, Scientific American, May 2001

[7] Jean-Claude-Laprie, Brian Randell, Carl Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, in *IEEE Transactions on Dependable & Secure Computing*. Vol. 1, No. 1, pp. 11-33.

[8] Tom Bellwood et al, “UDDI Version 3.0.2”, edited by Luc Clement et al, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[9] Sun Microsystems, "JAXR", http://java.sun.com/xml/jaxr/index.jsp

[10] Jeremy J. Panel, Jeff Z. Pan, "XML Schema Datatypes in RDF and OWL", http://www.w3.org/TR/swbp-xsch-datatypes/, 2005

[11] Chen Zhou, Liang-Tien Chia and Bu-Sung Lee, "DAML-QoS Ontology for Web Services", *ICWS*, pp472-479, 2004

[12] Glen Dobson, "Quality of Service in Service-Oriented Architectures”, http://digs.sourceforge.net/papers/qos.pdf

[13] E.M. Singh, M.P Maximilien, "A Framework and Ontology for Dynamic Web Services Selection", *IEEE Internet Computing* Vol. 8, No.5, pp84-93, 2005

[14] Ian Horrock et al, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", http://www.daml.org/2003/11/swrl/, 2003
"SWRL

[15] Jerry R. Hobbs, Feng Pan, "An Ontology of Time for the Semantic Web", 2004, *TALIP*, Vol. 3, No.1, pp.66-85, 2004