QoSOnt: A QoS Ontology for Service-Centric Systems

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

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

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

Abstract

This paper reports on the development of QoSOnt: an ontology for Quality of Service (QoS). Particular focus is given to its application in the field of service-centric systems. QoSOnt is being developed to promote consensus on QoS concepts, by providing a model which is generic enough for reuse across multiple domains. As well as the structure of the ontology itself, an example application currently in development, SQRM (Service QoS Requirements Matcher) is discussed. This application is used to highlight some of the advantages of the ontology. As well as standardisation, one such advantage is the ability to perform certain types of automated reasoning on QoSOnt and anything defined using QoSOnt.

1. Introduction

A service provider's business relies upon customers being able to trust the services they provide. There are many elements which contribute towards trust in this context, one of which is the availability of Quality of Service (QoS) information. In this paper the term QoS will be used to denote all non-functional aspects of a service which may be used by clients to judge service quality. This extends other more restrictive QoS definitions such as the common interpretation of QoS to mean network performance attributes. QoS data is of particular importance in service-based systems because services are generally black-box – being exposed purely through their WSDL interface. Moreover, in the situation where a service marketplace exists, quality will be traded off against cost by customers, making the judgment of service quality a key issue.

Simply having QoS data available is not quite enough in itself to achieve client trust. The provenance of the data is of utmost importance. In this paper, we assume that we have some trusted source of data, although we accept that achieving this is a non-trivial problem to which there is currently no complete solution.

As well as simply retrieving QoS data, customers may wish to query the data and select services based upon their QoS requirements. In order to enable the client, provider ant intermediaries to intercommunicate in order to achieve this, a common way of specifying QoS is needed. For this purpose we have designed QoSOnt - a QoS ontology.

The structure of the remainder of the paper is as follows: Section 2 examines the background technologies relevant to the project. Section 3 presents the QoSOnt architecture in detail. Section 4 examines the SQRM tool developed in parallel to QoSOnt. Section 5 examines the evaluation mechanisms for QoSOnt and SQRM. Finally section 6 provides information on future work and conclusions.

2. Background

This section briefly explains some of the technologies which form the background to QoSOnt: including SOA, Ontologies in general, OWL as an example ontology language and OWL-S, an OWL ontology for web services.

2.1. Service-Oriented Architectures

Service-Oriented Architectures (SOAs) are exemplified by the Web Services Architecture (WSA) and Open Grid Services Architecture (OGSA). One of the key selling points of these architectures is interoperability through the use of already ubiquitous technologies such as XML and HTTP.

SOAs are becoming increasingly popular with those who believe in the VO (Virtual Organisation) vision. VOs consist of an organisational structure rich in co-operation across structural, temporal and geographic boundaries. The flexibility to create, modify and destroy co-operative relationships without the cost and time lag seen in more rigid environments is paramount.

The development of DCOM, CORBA and more recently web services, has been driven primarily by the need for this improved business agility and greater emphasis on more dynamic service offerings. SOAs should be technology agnostic (not relying on given hardware, platforms, transport etc), and in doing so, eclipse the limitations of more rigid, costly interaction mechanisms built from technologies like EDI.

VO development depends on an expanding supply of services capable of interaction through common interfaces.

Diversity of services requires a number of different technologies to be developed. In order to compose services and in turn orchestrate them, workflow languages such as WSFL are required, enabling higher level service support for functions including fault tolerance. In order to discover potential services, discovery mechanisms like UDDI, UDDIe, WSIL, etc. need to be further developed to provide richer semantic information than they do at present.

The diversity of service instances also requires the implementation of higher level structures to support service monitoring, service negotiation, and the formation of legally binding documents (which could take the form of an informal SLA or service usage contract). As resource-based SOA instances commercialise, creating more dynamic services, it becomes increasingly difficult to maintain understanding between clients/services, for example, what is meant by a given term, in what way can it to used, in what domain does it exist, etc. It is these situations that have led to the increasing attention on ontologies in SOAs, and in particular on a QoS ontology for SOAs.

2.2. Ontologies in Software Engineering

In software engineering, an ontology can be defined as "*a specification of a* conceptualization" [1]. 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.

The boundaries between taxonomy and ontology, and between data modelling and ontology engineering can be confusing. In its simplest form an ontology *is* simply a taxonomy of domain terms, which in turn *is* clearly a form of data model. However, taxonomies are little use in machine reasoning. The term ontology also implies the addition of domain rules, used to aid machine reasoning.

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.

Problems that could be addressed through careful ontological design pervade much of our lives. Worldwide deaths are recorded through referencing to the ICD-10 WHO taxonomy [3]. The complexity of which should not be underestimated. The following code is given for a death involving a volcanic eruption whilst waterskiing in a public library:

X35.2.0

This is a good example of combinatorial explosion, in that any location, activity, and environment can be combined due to the lack of suitable ontological structure. Using an ontology would add, for example, the ability to constrain possible combinations of properties in order to create a more accurate model of the world.

The following section discusses OWL, which is an example of an ontology language. Since it is reasonably representative of such languages this should also give a useful overview of the constructs which generally make up an ontology.

2.3. OWL

OWL [4] is the Web Ontology Language, designed for publishing and sharing ontologies via the web. OWL is based upon the Resource Description Framework (RDF), which can also be regarded as a simple ontology language. Unlike OWL, RDF only provides limited support for concepts such as cardinality, datatyping, etc. RDF in turn is built upon XML. There are three 'species' of OWL – but the most useful for reasoning - OWL-DL - corresponds to a description logic. [5] gives a good introduction as to what this means.

OWL provides the base constructs for building an ontology. The two most important are called Class and Property. Other common names used for Class elsewhere include concept, category and type; whilst Properties may be referred to elsewhere as slots or relations.

The majority of the other OWL constructs exist to allow Classes to be defined. Defining a Class consists of precisely stating the requirements for individuals to be members of that class. A class definition is therefore synonymous with the set of all individuals meeting its membership requirements. A key feature of OWL and other description logics is that subsumption relationships can be automatically computed by a reasoner.

The following snippet from QoSOnt 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, UnmeasurableAttribute) could be defined anywhere in the file. Editing OWL manually can be difficult for the same reason. We used Protégé [6] and its OWL plug-in in our development of QoSOnt.

Part of the reason (other than avoiding ambiguity) for the relative complexity of this definition is that OWL works under an open world assumption. This means that no assumptions are made about anything unless they are explicitly stated or can be inferred from asserted facts. Open world reasoning is often counter-intuitive to those used to closed-world data modelling, and is not appropriate in all situations. OWL is extremely difficult to debug by hand due partly to its verbosity, making visualisation tools such as Protégé an essential part of ontology development.

2.4. OWL-S

OWL-S [7] is an OWL ontology for describing web 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" [10]

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. How to link the OWL-S and QoSOnt ontologies for the purposes of service QoS description is touched upon in Section 3.

3. The QoSOnt Ontology

QoSOnt was developed by a process of examining existing QoS specification languages ([8], [9]). The majority of detail in QoSOnt is in our specific area of interest, which is dependability, where we have built upon existing work by making use of an existing taxonomy and modelling commonly used metrics.

QoSOnt represents many of the commonalities discovered between the QoS specification languages examined. Unlike most of these languages however, QoSOnt also aims to be generic enough to be used no matter what one's particular view of QoS is. Our approach has been to provide a base set of useful constructs which cover common cases. These also exist as an example to others who wish to model their own QoS viewpoint on top of the basic QoSOnt Classes.

To facilitate reusability and extensibility the ontology has been designed from the beginning to be modular in nature. Each "module" is an ontology in itself. Ontologies in higher layers will specialise and build upon those from lower layers. The ontologies fall into three layers as shown in Figure 1.

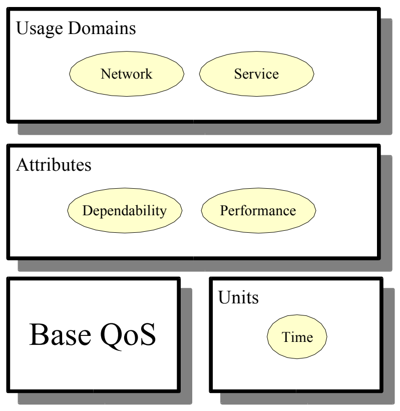


Figure 1. Layers of the ontology

The architecture is designed to allow third parties to replace parts of the ontology as needed. For instance they may have a different view of dependability to ours, or have produced a time ontology which suits their purposes better. Obviously this is only useful if the relevant ontologies are shared with the community they wish to interact with.

The base QoS layer contains generic concepts relevant to QoS. Unit ontologies also logically reside in this layer. Time is the most relevant unit in QoS, and is therefore the only unit ontology defined at the moment. It represents units of time and how to convert between them. This means that an inference engine could establish, for instance, that 1 minute is the same as 60,000 microseconds. This is particularly useful if clients use the same metric as providers - but different units.

The attribute layer contains ontologies defining particular QoS attributes and their metrics. On top of this is the domain-specific layer, which links the lower layers to specific types of computer system. For instance, the network ontology defines that certain QoS attributes are specific to a particular network route and the service ontology that QoS attributes sometimes refer to particular services, service operations, etc.

In the following sections we discuss each of the layers of QoSOnt.

3.1. The Base QoS Ontology

The base QoS ontology represents a minimal set of generic QoS concepts (as shown in Figure 2). We introduce the concept of a QoS attribute, and its unmeasurable and measurable subclasses. In using the ontology it is entirely optional whether one chooses to use these sub-classes or create one's own. Ontologies allow multiple inheritance, so many different classifications are possible. Indeed, a QoSOnt specification will always subclass something from the attribute layer as well as from the domain layer (e.g. to say that what is being referred to is the attribute reliability, and it is specifically the reliability of service X).

Unmeasurable in this context relates to attributes which cannot be measured from a given viewpoint. An example of this could be adherence to a particular standard. Anything which is measurable has a metric (as the OWL in Section 2.3. shows). A metric represents one way of measuring a specific QoS attribute. 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.

Measurable attributes have one or more associated metrics. At this level in the architecture we do not prescribe individual metrics; these are defined in more specific attribute ontologies. We define a metric 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.

A "physical quantity" has one or more associated "units". In many cases a numerical value alone cannot be understood without its unit type (e.g. You need to know whether "time to complete" is quoted in seconds, microseconds, milliseconds, etc.). Many metrics in QoS involve time, which is why we have included the time – ontology in QoSOnt. Other types of physical quantity are rare in QoS – but the structures are there to model them when they are required. Percentages can also be modelled as units in QoSOnt. This gives the ability to understand that availability of 0.99 is the same as availability of 99% for instance.

For metrics which have values with simple types (e.g. alphanumeric strings or integer counts) a new datatype property would be included in that sub-class of Metric.

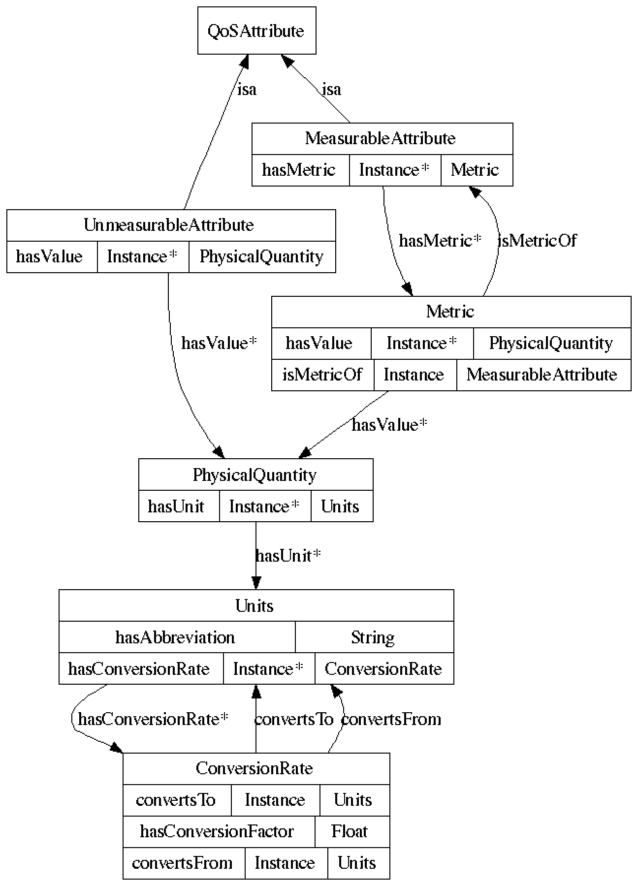

QoSAttribute
isa
isa
MeasurableAttribute
hasMetric | Instance* | Metric
UnmeasurableAttribute
hasValue | Instance* | PhysicalQuantity
hasMetric*
isMetricOf
Metric
hasValue | Instance* | PhysicalQuantity
isMetricOf | Instance | MeasurableAttribute
hasValue*
hasValue*
PhysicalQuantity
hasUnit | Instance* | Units
hasUnit*
Units
hasAbbreviation | String
hasConversionRate | Instance* | ConversionRate
hasConversionRate*
convertsTo
convertsFrom
ConversionRate
convertsTo | Instance | Units
hasConversionFactor | Float
convertsFrom | Instance | Units

Figure 2. The base QoS ontology

Note that Figure 2 shows the Properties and Classes of the ontology, but one may question how the domain rules talked about in Section 2.2. are modelled. In OWL (and therefore QoSOnt), it is the Class definitions which constitute the domain rules. These definitions not only define the Class in terms of necessary and sufficient conditions for membership, they also constrain the use of Properties and therefore the interrelationships between classes.

These Class definitions consist of OWL constructs such as those detailed for MeasurableAttribute in Section 2.3. There is no succinct way of representing these. For the sake of QoSOnt most Classes are defined in a pragmatic manner. That is, they are defined with sufficient rigour to be identified within the confines of QoSOnt (e.g. a MeasurableAttribute is any QoSAttribute with a hasMetric property which is a Metric). However, philosophical questions of the defining nature of the concepts modelled have not been considered. Since the ontology is an engineering artefact this is deemed to be a sensible approach.

3.2. The QoSOnt Attribute Layer

Figure 3 shows some attributes from both the dependability and performance ontologies (prefixed by d: and p: respectively).

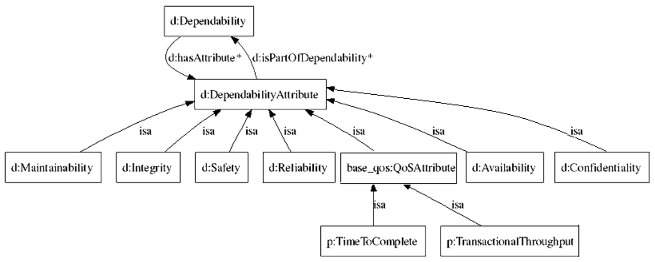


Figure 3. Example Classes from the attribute layer

The full dependability ontology is largely based upon the taxonomy defined in [11]. It not only includes the ability to represent dependability attributes – but also means of achieving dependability and dependability threats. These latter may be of less relevance to QoS – but will find use in other forms of specification. There is therefore an overarching concept of dependability, as shown in Figure 3.

The ability, given a particular Metric, to find the QoS attribute it measures through the *isMetricOf* Property (see Figure 2) allows other Metrics for that attribute to be found. With the overarching concept of Dependability, a further step may be made through the *isPartofDependability* Property. This Property gives access to the Dependability Class and therefore all information relevant to dependability (threats, means, etc.). This shows how the attribute layer provides a good point at which to provide hooks to non-QoS concepts. Doing so allows the integration of ontologies for further types of system description.

As well as the reusable dependability (or other attribute) ontology, a further ontology contains actual metrics (e.g. probability of failure on demand, mean time between failures, mean availability, etc.). It is this level of detail which is perhaps the most important; as it is only once specific metrics are added that QoSOnt is usable for QoS specification.

3.3. The QoSOnt Usage Domain Layer

Ontologies in the usage domain layer link QoS to a particular class of system. Currently QoSOnt supports networks and services as types of system that QoS may refer to. As with all layers, this can easily be expanded upon.

The most important part of the service ontology simply links the concept of "QoS attribute" and "service". Since we are working in the web services arena we use the Service class from the OWL-S ontology. Our ontology can also enhance the OWL-S ontology by providing concrete Classes to act as its "ServiceParameters". An example is shown in Figure 4.

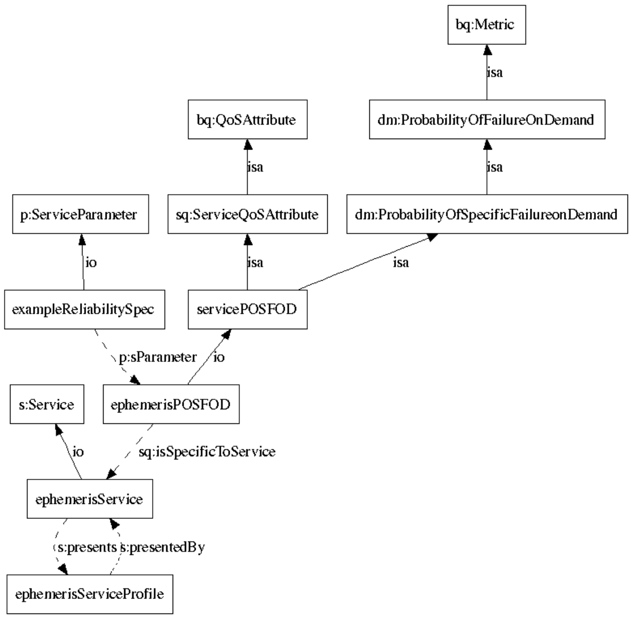


Figure 4. Example of the QoSOnt/OWL-S link

The prefixes in the figure refer to the following namespaces: s: OWL-S Service, p :OWL-S Profile, bqos: QoSOnt Base QOS, sqos: QoSOnt Service QoS, dm: DIGS metrics (Which contains our dependability metrics in the Attribute Layer). The solid lines labelled *io* indicate "instance of". The instances are the part which would actually be visible in a specification. The Service instance (ephemerisService in the figure) would be the point of entry to the OWL-S specification. The figure shows that a service presents a profile and that a profile has a parameter. Also depicted is how such a service parameter can make use of QoSOnt in this case by specifying a ServiceParameter which is Probability of Specific Failure on Demand. The unusual terminology is to distinguish it from probability of any failure on demand. "Specific Failure" refers to the fact that it represents the probability of one particular service failure defined using the dependability ontology.

As well as service-specific, certain QoS attributes are also operation-specific (e.g. time-to-complete, accuracy) and therefore reference the OperationRef class from OWL-S. Other attributes, e.g. reliability, may be best modelled as workflow specific since they are specific to a usage pattern. OWL-S provides a Process class which is much like a workflow. However it would be preferable to also be able to reference other types of workflow definition (e.g.

BPEL4WS).

A provider (or QoS measurement service) publishes QoS data as part of their OWL-S description or directly using QoSOnt alone. The former is the preferred option as it gives a standard way of referring to the service, operation, etc. in question. It also provides a single point of access for the complete specification.

The following section describes a tool which differentiates between comparable services by matching user requirements against published service QoSOnt specifications.

4. SQRM: A QoSOnt Application

To demonstrate the use of the ontology, and aid in its evaluation, a tool for service discovery, differentiation and selection based upon QoS requirement 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 cycle:

- Service Discovery
- Requirement Specification
- Service Querying – Differentiation

Beyond these stages the following external service interactions need to take place to complete the service cycle:

- Service Negotiation
- Service Agreement generation
- Service Monitoring (Encompassing mediation)

Though these are outside our current project scope, these three areas are also likely to make use of QoSOnt, as they are likely to reference many of the QoS attributes used in service discovery and differentiation. The following sections give an introduction to the different parts of the tool and their relationship to QoSOnt.

4.1. Service Discovery

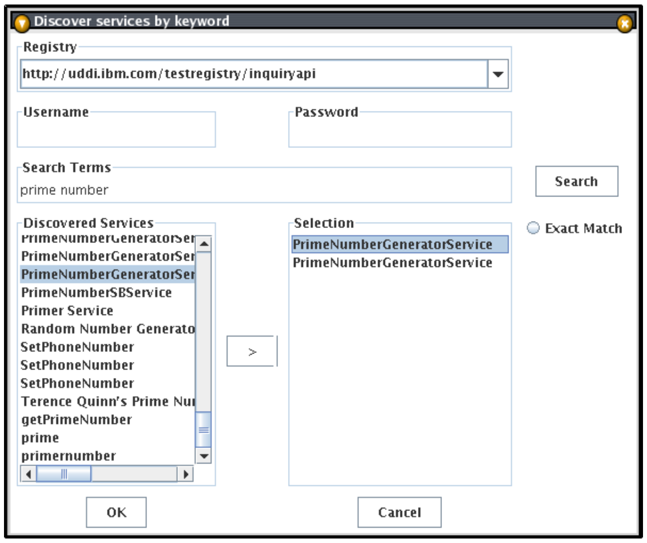
Service discovery in SQRM consists of querying a UDDI registry. This consists of a keyword search (as shown in Figure 5). The user can discover broadly similar services (or functionally identical services by examining the WSDL).

A feature planned for the near future is the ability to search by TModel. A TModel in UDDI is a generic data structure which provides *"...the ability to describe compliance with specifications, concepts, or even shared design..."* [12]. A UDDI entry for a particular service may have many associated TModels.

One use of a TModel is to specify adherence to a common WSDL interface. The originator of an interface can publish a TModel via UDDI which others can then use to show they adhere to the same interface. A "Type category" called wsdlSpec in the UDDI specification is used to indicate exactly such a use of TModel. By allowing SQRM users to search for services advertising a particular wsdlSpec TModel we provide a means to automatically retrieve a list of functionally identical services for comparison. SQRM will then gain access to the QoS data for a service by retrieving the URI for the data from the TModel in question.

In the same way, a TModel for QoSOnt or OWL-S (of the specification category) can be published in UDDI to allow providers to state that their services offer a specification adhering to either of these ontologies.

SQRM expects to find one of these TModels. A TModel must indicate, among other things, the URI through which to retrieve the specification in question. SQRM uses this to gain access to the QoS data for a service. Note that the QoSOnt specification that the URI points to may, in practice, be provided by a third party.



http://uddi.ibm.com/testregistry/inquiryapi

prime number

PrimeNumberGeneratorSer
PrimeNumberGeneratorSer
PrimeNumberGeneratorSer
PrimeNumberSBService
Primer Service
Random Number Generato
SetPhoneNumber
SetPhoneNumber
SetPhoneNumber
Terence Quinn's Prime Nu
getPrimeNumber
prime
primernumber

PrimeNumberGeneratorService
PrimeNumberGeneratorService

Figure 5. Service discovery in SQRM

4.2. 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 what forms 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, ranging from governments to international collaborative organisations including the World Health Organisation (WHO), and individual research institutions and universities. Different algorithms can be used to analyse data and the time taken for the analysis process makes QoS of great importance.

Some techniques for analysis of epidemiology are more applicable to some situations than others, and may depend on the time available for computation. For example, for a given sample size the use of a MCMC (Markov Chain Monte Carlo) computation may yield unacceptably inaccurate results if not run for a large enough quantity of time. For other algorithms perhaps a final result is attainable, and cutting computation short would render results useless. Information of this type can be built into an ontology, creating a rich information resource.

Our scenario pictures a situation where; given a number of possible services, capable of supplying a number of different formula computations; service differentiation and requirement matching tools play an important role.

Client software for analysing epidemiology data could potentially have a number of different QoS requirements. For example, the requirement of certain degrees of accuracy dependent on the algorithm used for analysis i.e.:

Algorithm A Accuracy > S Using Metric N
or
Algorithm B Accuracy < P Using Metric M

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 user 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". QoSOnt alone is sufficient to express these. If a provider wishes to use more complex forms of proposition then the same hybrid approach as for requirements may be used. For instance, the XML schema includes the ability to represent service classes (i.e. set offerings of the same service with different service levels). This is not currently possible to do with QoSOnt, although it is something which is likely to be incorporated in the future. The provider may also wish to express the inherent interdependence of certain metrics/attributes (e.g. server load and transactional throughput).

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. Figure 6 illustrates this with a screenshot of a requirement expression as it is created (MTTC is an abbreviation for Mean Time to Complete). QoSOnt allows constraints on the Values a Metric can support to be taken into account dynamically as the expression is constructed.

The expression shown in the figure states the following requirement: *((MTTC<500 milliseconds) AND (Mean Availability>0.9)) OR ((MTTC<1 second) AND (Mean Availability>0.99))*. This shows how trade-offs can be expressed: That time to complete can be traded off in favour of availability.

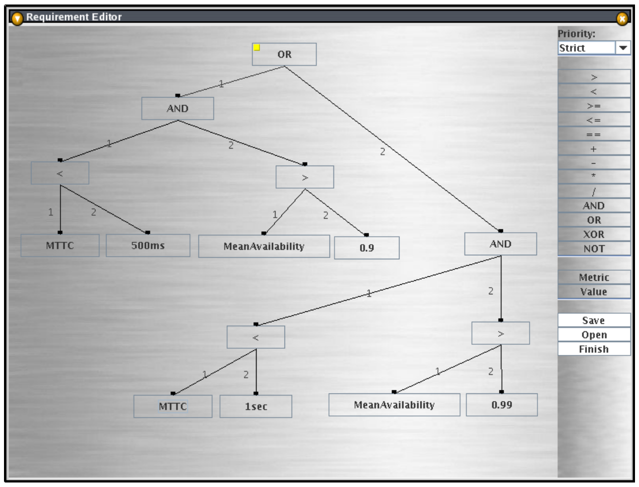


Figure 6. Requirement construction

4.3. Requirements Matching

Requirements matching is a bottom up evaluation process, which starts from the leaves. That is, for each service being considered, the asserted values of Metrics can first be established from the published capabilities. The truth value (or arithmetic result) these produce in the parent sub-tree can then be established, and this can continue to filter up until a truth value is established for the whole tree.

As an example of the matching process consider the requirement shown in Figure 6. If MTTC for a service under consideration is found to have a published value of 600ms then the left hand sub-tree immediately evaluates to false. This is because plugging a value of 600ms into the MTTC leaf makes its immediate parent (<) evaluate to false. Since its grandparent is an AND operator no further evaluation is needed to assign false to the left sub-tree rooted at AND. On the other hand, the opposite is true of the right sub-tree. Plugging 600ms in (and using QoSOnt to convert the units) makes > evaluate to true, the second child of the AND sub-tree therefore also needs to be evaluated. If MeanAvailability is found to be >0.99 then the whole requirement will be met and "true" will propagate up to the root of the tree.

Where the requirement is strict (i.e. it "must" hold true) this provides a simple yes or no match. It will also be more efficient as not all terms will always need to be evaluated (as for the left sub-tree in the above example). If insufficient information is available to make a conclusion this must also be taken as a non-match. For instance, if no specification for MeanAvailability was provided for a service then no truth value could be assigned.

Specifying the requirement as non-strict will allow for a more detailed comparison than simply matching or not matching. Non-strict matching might be useful, for instance, when a lot of required data is missing or if no strict match can be found. However, if the requirement is non-strict then there are a number of issues as to how to judge how well a particular service matches it. At the time of writing this is not a problem we have implemented a complete solution to. The approach we propose involves assigning a score based upon the level of match achieved. A lower score would have more required terms which evaluate to false or are missing. The score filtered up to the root of the tree could then be used to assess the relative suitability of services.

The ability to create QoS requirements involves understanding the underlying meaning of the attributes and their metrics. QoSOnt can supply much of the information needed for human inspection as well as providing UI constraints to avoid misuse of terms. For example, acceptability direction for a given metric (is high or low better), unit type, and so on. For non-strict matching we hope to produce advice and warnings based upon the semantics of the requirements created as well as an indication of how well the services match.

5. Evaluation

The evaluation of an ontology such as QoSOnt ultimately relies upon its application by the research community. 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 in the form of the epidemiology example in Section 4.2.

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.

SQRM's implementation has given concrete examples of QoSOnt's use by the client for service differentiation, by the provider for publishing QoS data, as well as by intermediate software in the matching process. Whilst there are extensions we wish to make in terms of the metrics we have modelled in QoSOnt, we have found that QoSont has not restricted us in modelling those we have already considered. However, to reduce the work required to model new Metrics, the possibility of providing some generic base Metrics has arisen.

We also 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.

6. Conclusion & Future Work

In conclusion, this paper has put forward a workable QoS ontology, outlining its objectives with reference to the service cycle as a whole, and specifying both its overall design and implementation.

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.

In terms of developing SQRM there are many user interface enhancements which we are considering, including, among other things, adding the ability to check the availability of services upfront; a wizard for requirements creation; and a visualization of the matching process so that non-strict cases can be judged by user.

7. References

[1] T. R. Gruber, "A translation approach to portable ontologies", *Knowledge Acquisition*, 5(2):199-220, 1993, http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html

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

[3] ICD-10 WHO Ontology, http://www.who.int/classifications/icd/en/

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

[5] 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/

[6] The Protégé Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu/

[7] DAML, "DAML Services", http://www.daml.org/services/owl-s/

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

[9] Glen Dobson, Russell Lock, http://wiki.nesc.ac.uk/read/pa9?ParametersOfQoS

[10] Tim Berners-Lee, James Hendler, Ora Lassila, "The Semantic Web", Scientific American, May 2001

[11] 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.

[12] 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