Developing an Ontology for QoS

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

ABSTRACT
This paper examines the development of an ontology for Quality of Service (QoS). This ontology is being developed to promote consensus on QoS concepts by providing a model which is generic enough for reuse across many domains. Our specific application is to the domain of service-based systems, and we have a particular interest in the QoS attributes which are part of dependability. A further emphasis of this paper is the way in which this work relates to the DIRC research themes. The main theme addressed by the paper is structure, though the applications of the ontology we aim to develop also touch upon risk, diversity and timeliness.

Keywords
Structure, ontologies, QoS

1. INTRODUCTION
In computing, 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:

- They facilitate interoperability.
- They facilitate machine reasoning.

Ontologies are already used to aid research in a number of fields. One of the more interesting to DIRC 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.

Ontology use pervades much of our daily lives, and deaths. Deaths are recorded through correct referencing to the ICD-10 WHO ontology [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

There is a degree of confusion between a taxonomy and an ontology, and between data modelling versus ontology engineering. In its simplest form an ontology *is* simply a taxonomy of domain terms, which in turn *is* clearly a form of data model. Such a taxonomy aids interoperability, but does little to aid machine reasoning.

Generally, the term ontology infers that domain rules as well as terminology is modelled. As such rules are added to the taxonomic classes, the data model becomes more likely to be called an ontology. However, the choice of formal representation depends on exactly what is to be represented. It may therefore be the case that something called a data modelling language may prove to be more suitable than an ontology language for representing certain conceptualisations (particularly those which are largely taxonomical). For a general introduction to the common aspects of ontological construction see [4].

This paper describes the development of an ontology for QoS. QoS research aims to allow the clear statement of non-functional requirements; reasoning about such requirements during design; as well as specification, monitoring, negotiation, and provision of the relevant levels of service once the system is deployed.

Ontologies are designed to make the standardisation of terms for given domains, simpler and easier to achieve. Standardisation of terms and structure has been a considerable stumbling block in the development of many QoS specification projects; often relying on proprietary language formats, that rarely become popular enough to provide a complete vocabulary of terms, or mechanisms for translation between different interpretations. The idea of a QoS ontology for use across technologies is therefore an appealing one. For more information on existing QoS specification languages see [5].

The structure of the remainder of this report is as follows: Firstly, a discussion of the motivation for a QoS ontology. Secondly, an examination of the QoS ontology itself, including an introduction to the hierarchies we are currently in the process of designing. Thirdly, a look at the way in which the DIRC themes of Risk, Structure, Timeliness, Diversity and Responsibility impact on the creation of such constructs. Finally, a brief statement outlining our current state of development, and the potential for future work.

2. MOTIVATION FOR A QOS ONTOLOGY
An ontology is not in itself an end, but rather a means to many ends. We are seeking to provide ourselves (and hopefully, in the future, others) with a unifying base on which to build QoS sub-systems. Our main aim is therefore standardisation. The use of a standard itself we see as a simple component of a "structure" for dependability. Moreover, our explicit model of QoS (and specifically dependability) concepts is something which is useful in the formal discussion of "structure" and the effects of different structures. The ontology can be easily extended with new concepts and existing reasoning tools can be used to highlight ambiguity or inconsistency.

The main pitfall we seek to avoid is straightforward disagreement or misunderstanding of terms. Taking availability as an example QoS attribute can demonstrate the extent of this problem.

Availability is generally represented as a probability that a system is responsive at a randomly selected point in time, i.e. It is essentially a mean of responsiveness over the lifetime of the system (or some period which is agreed to be representative of system lifetime).

However, such information is limited in its usefulness. Statistical summaries by their very nature hide information. An availability of 96.6'% may equally refer to regular downtime of 2 seconds per minute or one day per month. Since system usage will often fall into a particular temporal pattern this means that the mean availability will often be misleading. For instance the 96.6'% availability figure may represent regular maintenance which always takes place on the last Sunday of the month. If the user in question never uses the system at a weekend then this downtime is irrelevant to them. In practice, they may find it actually has better availability than they expected.

Other possible representations of availability therefore include a complete downtime history or a listing of regular downtimes.

All of these representations of availability (and others) are useful in certain situations. An ontology should therefore allow their use and make explicit their interrelation whilst avoiding ambiguity or confusion.

Similarly, a number of different reliability metrics are commonly used in industry, but it cannot be argued for the purposes of this research project that any given metric is the correct one, or even most preferable. Instead a number of possible metrics are put forward, with the proviso that a given interaction may involve, all, some or none of them dependant on the situation.

- POFOD (Probability of failure on demand)
- ROCOF (Rate of failure Occurrence)
- MTTF (Mean time to failure)
- MTTR (Mean time to repair)
- MTBF (Mean time between failures. MTTF + MTTR)

Generally speaking, reliability metrics are used to measure three main areas of system operation:

- Time to failure
- Time between failures
- Time to restart

Note: Time does not necessarily flow in hours, minutes etc, but could be measured in transactions, etc.

In the interests of brevity we will not continue this discussion of QoS attributes and their representation further here. Instead please refer to [6].

As well as standardisation and unification of terms such as reliability and availability, a QoS ontology provides some other useful capabilities. For instance, even if a human misunderstands the usage of a given instance of availability a reasoner will spot the different uses.

Also, certain things which are not explicitly classified at all can be classified automatically, and therefore have the correct domain rules applied to them. This is unlikely to be particularly useful for QoS attributes – but could for instance be used to classify the faults which are returned by a fault reporting system.

3. THE QOS ONTOLOGY

To facilitate reusability and extensibility the ontology has been designed from the beginning to be modular in nature. Modules (i.e. Sub-ontologies) fall into three layers as shown in Figure 1.

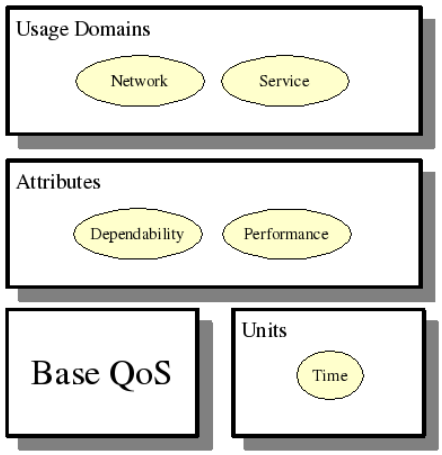

Figure 1. Layers of the Ontology

Some of our less well-defined sub-ontologies could perhaps be replaced with more complete third-party ones at a later date. The base QoS layer contains generic concepts relevant to QoS. It currently consists of a single sub-ontology. Alongside it sit unit ontologies. The only example unit sub-ontology defined at the moment defines 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.

The base ontology represents a minimal set of generic concepts (illustrated in Figure 2) – but is the most likely to be expanded upon as the ontology is put to use. Figure 2 shows the properties and classes of the sub-ontology – but not the logical rules which actually define a class in terms of necessary and sufficient conditions for membership.

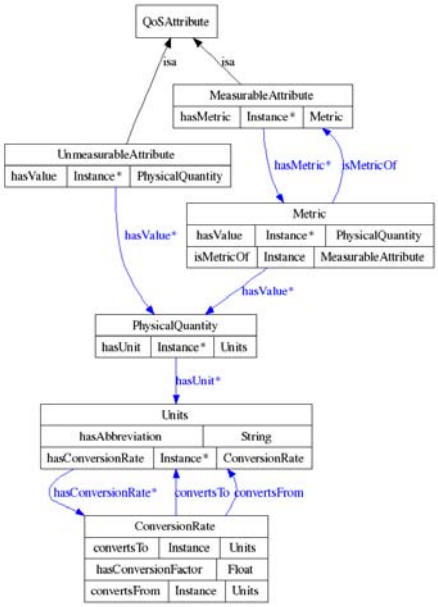

Figure 2. The Base QoS Sub-Ontology

We introduce the concept of a QoS attribute, and its un-measurable and measurable subclasses. In using the ontology it is entirely optional whether one chooses to use these sub-classes or create one's own. Un-measurable in this context relates to attributes which cannot necessarily be measured from a given viewpoint. An example of this could be adherence to the data protection act.

Measurable attributes have one or more associated metric. At this level in the architecture we do not prescribe what the individual metrics and formats are; these are defined in more specific attribute sub-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.) 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".

Figure 3 shows some attributes from both the dependability and performance sub-ontologies (prefixed by d: and p: respectively). The dependability sub-ontology is largely based upon the taxonomy defined in [7]. It not only includes attributes of dependability – 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. It is likely that such a class will later be defined for the concept of performance as well.

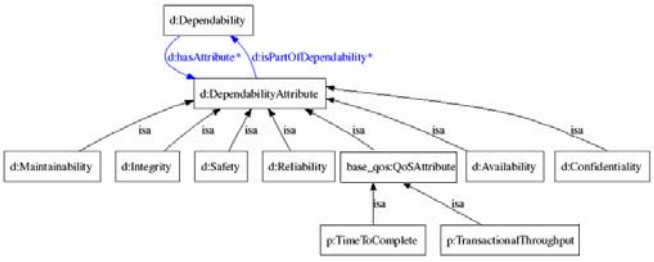

Figure 3. Example classes from the attribute layer

Some work is also yet to be done on relating security to dependability. There is no problem with attributes having multiple classifications in an ontology (so, for instance, confidentiality can subclass SecurityAttribute and DependabilityAttribute). We also need to model the fact that dependability attributes are inherently dependent on security. Once modelled, this interrelation could be used, for instance, to disallow the specification of availability without specifying the security attributes to avoid maliciously caused unavailability.

The most important part of the service sub-ontology simply links the concept of "QoS attribute" and "service". Since we are working in the web services arena we have encoded a version of the whole ontology in the OWL Web Ontology Language [8]. This means that we can reuse the Service class from OWL-S ontology [9], which is an existing ontology representing the service domain. Our ontology can also enhance the OWL-S ontology by providing concrete classes to act as its "ServiceParameters".

Certain QoS attributes are also operation-specific (e.g. time-to-complete, accuracy) and therefore reference the OperationRef class from OWL-S. For attributes such as reliability which are specific to a usage pattern, it may also be useful to reference a workflow in some cases. OWL-S provides a Process class which is much like a workflow. However it would be preferable to also reference other types of workflow definition (e.g. BPEL4WS).

On top of the sub-ontologies discussed above we are still in the process of extending the finer grained details of dependability attributes (essentially defining metrics). This is an ongoing process as we make more use of the ontology.

4. USING THE QOS ONTOLOGY IN SERVICE-ORIENTED ARCHITECTURES

We envisage the parameters populating the different ontologies to be used throughout the service cycle illustrated in fig 4. Service capabilities/client requirements, etc. could be expressed through the use of expressions made up of appropriate parameters. The following could be considered a simple example of an expression:

TimeToComplete < 1000ms

The attribute TimeToComplete (or more precisely one of its metrics) could then be referenced using the QoS ontology in order to reason about both its properties and relation to other attributes.

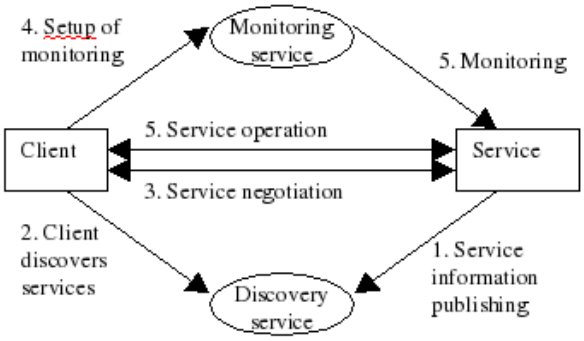

Figure 4. Service Usage Process

Figure 4 shows the sub-processes involved in service usage in a QoS-enabled architecture. These are: the provider publishing service information to a registry, followed by the client discovering the service via the registry (these are both supported by existing mechanisms such as UDDI). The third stage is negotiation. This may be a full negotiation of a Service Level Agreement (SLA) or it may, at its very simplest, represent an acceptance of published service specifications by the client. Having agreed on the service level to be provided, monitoring of the relevant attributes is set up, before the service is finally utilised by the client. Monitoring continues throughout service usage.

We envision the following as possible applications of our ontology in the service domain:

- Specification (at design time as well as at runtime, e.g. specifying the results of monitoring, client testing or provider capability claims)
- Requirements specification as input to service differentiation and selection
- Fault reporting
- SLAs and their negotiation
- Reasoning about QoS of composed services

The advantages of an ontology over some other standard for these purposes is the possibility for machine reasoning. This allows inferences such as that dependability is inherently linked to security, that end-to-end availability can be no greater than network availability for a particular route, that time-to-complete of 1 second is the same as that of 1000 milliseconds, etc.

An ontology can also infer classifications of objects. It might therefore be useful as the back-end of a fault reporting system. Faults reported through a standard form could be automatically classified as well as compared against existing fault instances already known, to see if the reported fault has already been identified.

The fact that the ontology itself can be extended (as well as having instances added to it, forming what is essentially a knowledge base) means that new inferences can be achieved as new knowledge representations are added.

5. THINKING THEMES

Section 2 briefly discussed how a QoS ontology contributes to DIRC's "structure" research theme. It was suggested that the ontology itself was a structural component for building more dependable QoS-enabled systems. By simply using the ontology as their base QoS representation, a developer, will avoid certain pitfalls common in the QoS domain. In this context the ontology has a similar function as a design pattern, in that it embodies practically gained knowledge in a reusable way. In its current state this is almost certainly over-stating the ontology's power. However, after a number of iterations of feedback from practitioners (including ourselves), and subsequent improvements we believe that this will be a realistic claim.

The ontology is also a useful means of discussing "structure" and its effects on system dependability. The ontology covers dependability means, threats and attributes as discussed in [7] _ providing a way to represent the interrelationship of these concepts. In the system development process, it also minimises ambiguity and misunderstanding between stakeholders, thus minimising design faults in the sub-systems designed specifically for dependability.

Our applications of the ontology in Service-Oriented Architectures also touch upon issues relating to other DIRC themes. Our "performance" attribute sub-ontology and "time" units sub-ontology obviously have some relation to the "timeliness" theme. As it stands this relation does not go very deep. However, one area we are looking to use the ontology in is in choosing between service compositions which perform the same task. Essentially, our ontology will give the ability to combine QoS data (including timing and performance data) with a workflow. We aim to model how various QoS metrics aggregate when services are composed and produce overall QoS values for the workflow/s. In the service world it will not be possible to achieve any strict guarantees – but this will at least give us the ability to e.g. choose the composition likely to complete most quickly.

Most services based on an economic footing require temporal constraints. The ability of services to state the timeliness of operations is a vital component in the service cycle. These could be applied in a number of different areas including:

- Process time available (Scheduling / load balancing)
- Accurate monitoring of the time taken in a request
- Constraints and capabilities expressed in service contracts / SLAs

Composing services from many sources (and potentially having further sources provide QoS measurements) also raises problems of "responsibility". One of our aims is therefore to ensure that the provenance of QoS data is carried with it.

A further possibility, if we continued to enhance existing service specification languages, might be to introduce the concept of a responsibility structure into the ontology. This could explicitly

model whether the party offering the composition takes complete responsibility, or whether responsibility remains with the atomic services, etc.

Another DIRC theme touched upon by composition is "diversity". The very idea of QoS in Service-Oriented Architectures relies on diverse service implementations being available. Since competing implementations would most likely offer much the same functionality they would compete by offering unique quality:price trade-offs. As well as providing a means of modelling QoS attributes of diverse implementations we also hope to provide other information, such as hardware platform, geographic location, network route, etc. all of which could be used to determine the most diverse service composition achievable. This information could, for instance, be used to choose the most suitable (i.e. diverse) component services for use in a fault tolerant container [10].

Most of the links between DIRC research themes and the ontology will only be fully realised through applying the ontology. The following section discusses some future work which will build upon the ontology.

6. THINKING THEMES

This paper details a work in progress. We have only outlined the base components and upper hierarchies needed to support them. We hope to continue populating the hierarchies we have created and to develop software tools suitable for the creation and manipulation of QoS specifications over the next few months. Initially we plan to work on requirements specification as input to service differentiation and selection.

The standardisation of attribute structure and QoS hierarchies is essential to the future adoption of more advanced negotiation and discovery mechanisms. The way in which attributes and their metrics are combined into expressions regarding QoS (e.g. assertions in a specification or requirement statements) is the logical next step to investigate. We are in the process of creating suitable user interfaces to allow the construction of expressions to differentiate between discovered services, and to specify service capabilities (regardless of how such capabilities are measured) as well as potentially to be used in the design process. In addition these constructs could be used to aid in the setup of monitoring services designed to prove adherence to a given agreement. The structure which emerges can be used to provide higher level services that support QoS throughout the operational envelope.

The structure of service operation ensures that services provide agreements to cover their liabilities and monitoring to ensure they do not breach them. The service cycle relies on both discovery, and monitoring structures to build trust into the process. Trust in turn reduces the risk inherent to the use of services with whom the client may not personally have dealt with before. The main aim of the standardisation process is to allow greater diversity of services, grounded through a common set of parameters.

Along with parallel work on reasoning about composed services the scope of this work is ambitious, but the ontology serves as a good base for this and other future work. Ideally, after serving our own pragmatic purposes, we can also take the ontology on to form the basis of a standard for QoS specification for use across the Web and Grid service community.

7. REFERENCES

[1] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993

[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] McGuiness D, "Ontologies come of age", http://www.ksl.stanford.edu/people/dlm/papers/ontologies-come-of-age-mit-press-(with-citation).htm

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

[6] Glen Dobson & Russell Lock, "QoS specification in service centric systems", http://wiki.nesc.ac.uk/read/pa9?ParametersOfQoS

[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] "OWL Web Ontology Language Reference", http://www.w3.org/TR/owl-ref/

[9] "OWL-S", http://www.daml.org/services/owl-s/

[10] Glen Dobson, Stephen Hall, Ian Sommerville, "A Container-Based Approach to Fault Tolerance in Service-Oriented Architectures", http://digs.sourceforge.net/papers/2005-icse-paper.pdf