Quality of Service in Service-Oriented Architectures

Glen Dobson

28th September 2004

Abstract

It is noted that there are two ways in which the term Quality of Service (QoS) is commonly used: firstly in its traditional distributed multimedia sense, which relates almost entirely to network performance, and secondly in its service-centric sense which refers to a wider range of non-functional service characteristics. In this paper the complexity of enabling this latter QoS definition in service-centric systems is discussed. A survey of existing technologies which are applicable in this area is undertaken, concentrating on QoS specification. From this survey it is concluded that there are a number of areas in which these technologies lack, or that remain open for research. In particular, the following needs are identified: a general purpose QoS specification language; a QoS-based service discovery mechanism to complement UDDI; a system for Service Level Agreement (SLA) negotiation, which should work with specifications in the aforementioned language; and systems for QoS measurement and monitoring. Research into using underlying technologies to provide QoS in service-centric systems and into mapping from SLA specifications to configuration of such technologies is also identified as key.

1 Introduction

Software developers spend the majority of their time concentrating on the functional aspects of the software they produce. It is, after all, their job to make the software they produce do the things that it is supposed to. However, it is often at least as important to the user that software not only provides the functions they require - but that, in its operation, it meets certain other non-functional requirements. For instance: speed of operation may be highly important in systems which require a rapid rate of transactional throughput; availability may be important in critical systems; security may be important where private data is being exchanged. In fact, it is almost always the case that a number of such factors are relevant to any one system.

The web service and Grid service architectures provide an appealing model in which to make use of third party software or software components (exposed as services). In using these architectures the need to be able to reason about the kind of non-functional factors mentioned above is highlighted because the user or developer has no direct control over these factors themselves. In service-oriented architectures it is therefore desirable to be able to clearly state non-functional requirements, to reason about them during design, and to be able to specify, monitor and negotiate the relevant parameters once the software or system is deployed. This allows the user or developer to more easily trust services, to choose services dynamically and to assess how much to pay for them (it is therefore one key building block in the creation of a service marketplace). Research aimed at achieving this vision groups the above concepts together under the banner of Quality of Service (QoS).

A large body of existing research already exists in the field of QoS as it is perceived from a networking, telecommunications and distributed multimedia perspective. As discussed later there is some overlap, but there is also a degree of disparity between this research and QoS research in the service-centric sense. In the following section this disparity is discussed and the meaning of Quality of Service is made more explicit by examining the way in which the term has traditionally been used and how it is being extended to better suit today's service-oriented architectures. In Section 3 the potential use of QoS in service-centric systems is discussed as well as how services

might be used in practice as this affects how QoS should be best applied. Section 4 discusses existing work on QoS, particularly in the area of QoS specification. The final section concludes by taking these technologies into consideration whilst discussing what is required of these and future technologies in order to achieve the service-centric QoS vision.

2 What does Quality of Service Actually Mean?

QoS refers to a level of service that is satisfactory to some user (i.e. it is not necessarily just about the 'best' level of service across the board - but about meeting user requirements). The related term Service Level Agreement (SLA) is often used to describe the agreement between the user and the service as to what constitutes 'satisfactory'. As well as QoS agreements, an SLA may also contain agreed cost, service functionality and other agreed parameters.

The term QoS, as it is most commonly used in practice, originated in the fields of networking, telecommunications and distributed multimedia. Even here the precise definition varies. However, it generally tends to refer to the request, specification, provision and negotiation of some or all of the following network characteristics:

- Bandwidth (or throughput)
- Latency (or delay)
- Jitter
- Error Rate
- Availability (or uptime)
- Network Security

These properties are often further subdivided (e.g. latency into one-way or two-way, error rate into packet loss or sequence error rate). Various metrics may also be employed to quantify these properties. For instance, jitter metrics might include the maximum difference in latency, the standard deviation of latency or some qualitative measure of the degree of variability in latency.

This concentration on network QoS appears to provide rather too narrow a definition of the term. Users are generally interested in the end-to-end QoS provided whilst using a specific service or piece of software. This means that not only does the network have an effect - but the software itself, the host system and even the user system may influence the QoS that the user will actually experience. Even in the realm of distributed multimedia this is the case (consider, for instance, the effects of using different codecs on the observed quality or of using a mobile device with limited resources for playback rather than a desktop PC).

Following the trend set in the area of service-oriented architectures, it is suggested here that the term QoS be used in a broader end-to-end sense, encompassing anything on the path from the user to the service which may affect quality. QoS may then refer to characteristics of the service host, the service implementation itself, the intervening network and the client system.

Some aspects of application or service QoS which may be of interest broadly fall into the following areas (as with network QoS, there are various metrics and sub-categories which may be applied to all of these):

- Dependability (including availability, reliability, security and safety)
- Accuracy of operation
- Speed of operation
- Failure Semantics

These characteristics are specific to the service, and hold some meaning no matter who the client is. Conversely, network performance characteristics vary depending on the route between the client and the service. Network QoS measurements taken between one client and some service can therefore not be used as a reliable indicator of what network QoS it is possible for that service to provide to some other client.

The characteristics of interest in the service host and the client system are largely the same. These broadly fall into the following categories:

- Dependability (including availability, reliability, security and safety)
- Speed
- Storage
- Operating System
- Installed Software
- Software Configuration

It may be that QoS parameters at all of the levels mentioned above need to be taken into account. For instance imagine some service operation that performs a calculation and returns the result to the client. If time to complete is of importance to the client then not only the service's speed of operation (e.g. mean time to complete) is relevant - but it may not be possible to ignore the network latency, jitter, bandwidth and error rate. To give the entire end-to-end picture the client system speed and scheduling policy may also need to be taken into account. The other reason client system characteristics may sometimes be of interest is if they occasionally affect the contract it is possible for a service to agree to. For instance the service provider may want to ensure you have a certain software configuration (e.g. a certain codec installed) before making any agreement.

The number of factors which could affect end-to-end QoS for an arbitrary service could no doubt be extended even further, and with the broadening of the scope of what a service actually is (beyond multimedia applications) it is often the case that network performance diminishes in its importance to the client compared to other factors. It is therefore worth noting here that a QoS language should be extensible in order to be applicable to new situations. It is also worth noting that in handling QoS specifications, descriptions and agreements that they may consist of any subset of the parameters mentioned here and may also involve custom parameters.

3 Quality of Service in Service-Oriented Architectures

3.1 Travel Agent Scenario

In this section the widely discussed 'planning a trip' scenario is discussed to give a point of reference for the following sections. In this case it is envisaged as a travel agent application. The aim of this application is to take the place of the human travel agent, allowing the user to make an informed choice and book a complete holiday package. This application might be created with functions to allow querying and booking of flights, hotels, hire cars and airport taxis. It might include weather reports, location and hotel reviews and currency ordering. All of these would be implemented by composing appropriate services in the client application.

It is probable that querying the booking and currency services would be free as the profit would come from actual bookings or purchases. If this was the case, the results from various providers could be aggregated (selecting as many services as necessary to give the maximum coverage). On the other hand, if these services were not free then their cost would limit the scope of search aggregation. Note that it is unlikely that providers will give QoS agreements for freely provided services (and this is perhaps one motivation for making such service free in the first place). If any agreement could be made then keeping search time to a minimum and agreeing good levels of availability would be of prime concern.

Speed of booking would be another key requirement from the client. It is likely that in the negotiation of the SLA the provider would add a further clause to this, agreeing to provide response times within some bound - but only when its server load is below some value. This highlights the need for third party QoS measurement and monitoring of SLA compliance (see the Measurement and Monitoring section) as the client cannot trust the provider to accurately report its server load in this case.

An agreed privacy policy and agreed levels of security would also need to be reached for each booking/buying service. As well as these familiar QoS parameters, it is also possible that the client may want to ensure other qualitative aspects of a provider's booking service such as levels of customer service (perhaps measured using some third party rating scheme), cancellation policy and so on.

The two main service architectures (the Web Services Architecture and the Open Grid Services Architecture) have up until now been used slightly differently. The first has been applied mainly in e-Business, whilst the second has been applied mainly in e-Science. This in no way represents their applicability in their current forms to either domain - but perhaps says something about their target audiences. The scenario discussed above highlights some QoS issues which may commonly occur in e-Business and e-Commerce applications. It is worth noting that different issues arise in e-Science, due to its more computational nature (e.g. accuracy, storage).

3.2 How Services Might be Used in Practice

Before one envisages how QoS could be used in service-centric systems, how such systems would themselves be used must be considered. There are essentially three usage scenarios for services, differing in how the services are discovered. These scenarios must each be considered when envisioning how QoS may be enabled for services:

- The first scenario involves no automated service discovery at all: the client has trusted providers and goes to them for services that they are well known for providing; deciding by their own means that the balance of cost and quality is good. Despite the 'service vision' this situation may be quite common as it is closest to how we tend to use services (relatively statically) in the real world. For instance we will not search around for a hair dresser each time we need one - but go to one we have previously found acceptable until they fail to meet our needs. This is despite the fact that there may be 'better' hairdressers out there if we searched. This scenario also allows for a good trust relationship to base SLAs upon and for long term SLAs to be put in place, thus reducing the overheads involved in their negotiation.

- The second scenario involves service discovery at design-time. The client application still uses services in a static way - but the relevant services are found by the designer using a discovery mechanism such as UDDI. This differs from the first situation only in that services can still be chosen on their merits (according to the discovery mechanism used) provided their details are available at design time. Where clients have no existing relationships with providers, want consistency of service quality and/or want to develop long term agreements and trust relationships with providers, this is a likely situation.

- The third scenario involves service discovery at runtime. Thus, the cost and quality of all available services may be assessed each time the client application (or one of its functions) is used. This allows the greatest flexibility to achieve the best price, quality, or some trade off of the two. This highly dynamic way of using services suits itself most to 'commodity' type services: the most important characteristics being that there are a number of competing providers (using the same or similar service interfaces) to choose between and that their relative cost and/or quality changes over time. For instance, in the 'travel agent' scenario there are likely to be a number of currency ordering services with a fairly standard interface, each offering varying rates and physical delivery times. If there are not a number of competing services, then this usage strategy would have no advantage to the client and could potentially

have disadvantages in terms of dependability, cost and maintainability, especially given that any SLA will necessarily be short term if it is possible to negotiate a useful agreement at all.

3.3 A QoS/SLA Specification Language

Since they overlap in what they must express, the language in which the QoS requirements are expressed is likely to be the same as the language expressing the QoS capabilities and the QoS agreements in the SLA. Essentially, one specifies the desired QoS, one the best possible QoS that can be provided, and one the agreed middle ground.

Language elements beyond QoS specification will also be necessary in order to allow SLA negotiation. For instance the final SLA must include the agreed costs and cost model, as well as third party involvement (e.g. the measurement service, monitoring service and negotiation intermediary) and the the agreement lifetime.

The most appropriate form of representation for QoS specifications is open to debate and may vary from application to application. The tendency in existing work is to allow specification of bounds on quantitative QoS parameters. It is clear that qualitative values must also be expressible as many QoS parameters simply cannot be expressed numerically.

Also, not every requirement is expressible as a bound or set of bounds alone: the most common case being that the requirement on one parameter may depend upon the requirements on some other parameter. This manifests itself when the parameters are inherently related: for instance weaker requirements on certain parameters, such as time to complete, may be necessary under higher server loads (as discussed in Section 3.1). Equally, the inter-dependency of parameters may be a choice made by the client: for instance they may decide that time to complete and availability are both important to them - but that for higher availability they are willing to trade off some degree of operational speed. Furthermore, a single figure is often misinterpreted or simply irrelevant (see the section on QoS measurement below).

The interdependency of requirements on different parameters is expressed in some systems by allowing weighting of parameters. Such expressions are somewhat limited however, as the relationship between parameters as they vary over their full range is unlikely to stay the same. Once one of the parameters has reached a very high value there is little to gain by it being higher and therefore the relation between the two breaks down. Similarly neither should ever be too low no matter what happens to the other. The next evolutionary step is therefore to have n-dimensional graphs for n inter-related parameters. This would not be particularly practical as it would quickly become difficult to manage, especially if one considers that manual requirements matching may take part. A more flexible and generally applicable approach would be to add a logical layer to the language to allow arbitrary relations between requirements to be expressed.

3.4 QoS-based Service Discovery

Service discovery mechanisms act as an aid to requirements matching in service-centric systems. The input to a QoS-based discovery process is a set of QoS requirements. The job of the discovery mechanism (whether human or automated) is to match these requirements to a set of available services based upon the published service QoS capabilities.

Currently the WSDL service description makes no reference to QoS. However, it seems logical to publicise the functional and non-functional service characteristics together. Including QoS capabilities in the WSDL or a separate, but related document would also have the advantage that QoS capabilities can refer directly to elements of the service functionality (i.e. operations, parameters, etc. in the WSDL). For instance, in the 'travel agent' scenario, an operation called 'book' may be described in the WSDL for one of the hotel booking services, and it is to this operation that the agreed time to complete for bookings should relate.

In terms of automating the requirements matching (or QoS-based service discovery) process, UDDI has no standard mechanism to allow it. The best way to support publishing searchable QoS data in UDDI would be the tModel. The tModel is a fairly general purpose data structure that an organisation can use to describe aspects of its service/s. This may only allow fairly limited

searches however. In practice QoS enabling UDDI may be unnecessary as it seems that after a coarse grain search for matching services the actual WSDL (and QoS capabilities) should be examined more carefully to find the best functional and QoS-based match or matches. Service discovery and selection is likely to be (at least) a two step process.

3.5 QoS Measurement

Simply taking QoS capabilities that are advertised directly by a service at face value is not advisable. Service providers have a vested interest in overstating their capabilities. To avoid this problem, the actual QoS being provided ay any one time by services should be measured, monitored and published by a third-party measurement service. Ongoing QoS measurement could also provide more dynamic, up to date QoS data and allow for a better profile of QoS provision over time to be built up.

The measurement of QoS parameters using a single numerical metric is often misleading however. In particular, aggregates over time can lead to a lack of relevant information. For instance, a service may advertise availability of 99% (over its entire lifespan). If the client wishes to use the service at a regular time in which the service is always down or at greater risk (thus accounting for the majority of the 1% unavailability) , then this availability figure does not tell them all that they need to know. To fill this gap, the wider context in which the number is being used must also be expressible. This context could include, in this case, a profile of the parameter's variation over time or known down-times, but is likely to vary in its contents from parameter to parameter.

In contrast to availability, the profile over time is of no use for some QoS parameters. For instance a provider may have X amount of storage available at the current point in time. What it had available at any other time is irrelevant: a client simply wants to know that if it wants to reserve amount of storage Y then X>Y. To ensure that clients can use the QoS metric most relevant to themselves, based upon what they regard as the most relevant data and most relevant context, one possibility is that the measurement service could act as a data mine, allowing the client to construct their own metrics on the provided data.

As noted earlier, measurements of network performance (and therefore published network capabilities) have their own particular difficulties: because the metric is specific to the route between client and service the measurement will vary from client to client. This can also be viewed as a problem of not understanding the metric's full context. Simply stating the context (i.e. the route to which the measurement applies) is unlikely to be much help on its own in this case as relevant measurements for the route to the given client are unlikely to be available. Furthermore, they will vary over time with levels of network congestion. In practice some estimate would perhaps need to be made on a per client basis at the time of the service call. This could be done using some representative test communications over the client-service route. The Slowing down of service calls and knock-on effects of QoS from this approach would be potentially severe however. This observation applies to QoS in general. The effects of measuring QoS on the actual level of QoS provided must be taken into account in reported figures. A measurement service would have to seek to minimise these effects and report them accurately.

3.6 QoS Monitoring

Monitoring is related to measurement (and the two may indeed need to be combined). Services may be monitored to detect failures (and therefore availability measurement may be done at the same time). Format checking may be performed on returned data (e.g. accuracy checking, checksums, etc.) by comparing the returned data against some specified template, with no need to update a metric. This templatE concept can be extended so that arbitrary checks may be made upon returned data.

SLA compliance monitoring may also be done. This is a complementary (or in some cases alternative) to measurement as a way of avoiding overstated QoS capabilities: if an SLA is formed stating that if it is not complied with then the provider will receive no payment, then there is no advantage to the provider in claiming greater capabilities than they actually have. The SLA would

need to include details of an agreed third party compliance monitor to enable this situation. The need to make predictive or historical measurements is then somewhat removed (depending on the metrics used) as the compliance monitor simply needs to know if, at the current moment in time, the QoS being provided is as per the SLA. For instance, rather than measuring availability, the provider could simply go unpaid whilst unavailable, or have some financial penalty made against them (depending upon the terms of the SLA).

The third-party monitoring of SLA compliance could also lead to a rating showing the reliability of providers' agreements. This could be made available to clients along with other service description data, acting as a further pressure on providers to ensure they can meet the agreements they make. The client would be unlikely to use a service with a poor compliance rating whatever QoS capabilities it claimed.

3.7 SLA Negotiation

The aim of SLA negotiation (whether human or automated) is essentially to reach a Service Level Agreement based upon a client's QoS requirements and a provider's QoS capabilities. In practice, the inputs to negotiation will involve other information from both parties. Some of this they will keep hidden from each other, such as information regarding their negotiation strategy. The rest includes the non-QoS elements already discussed in the QoS/SLA Specification Language section. These include cost, cost model, agreement lifetime, as well as details of any measurement, monitoring, negotiation or other intermediaries.

The negotiation language or protocol would work with chunks of the SLA specification language - but does not necessarily bear relation to it in its structure. The chances are, that in order to exist in a service world it will also be XML-based however.

The SLA reached by negotiation should include details of the parties involved, including agreed third party monitoring and measurement services, details of payment and cost model adopted, agreed functional service details and agreed Quality of Service details as well as actions to take when the provider's QoS falls outside the agreement (including service failure).

It is only in the most dynamic service usage scenario, where the client's service providers change regularly, that SLAs are likely to be be renegotiated regularly. In this situation, automated service discovery and SLA negotiation are key to enabling the vision.

Even in this situation, long term SLAs are likely to hold a number of advantages. Considering the 'travel agent' scenario for instance: the agreements on speed of operation for the search services would be completely negated if operation calls were regularly preceded by a long winded discovery and negotiation process. Equally, agreements on availability would have no purpose if the service was discovered and was then to be used immediately and not reused again without first rediscovering it. So, even if the search services were to be selected dynamically at runtime, it may be worth having pre-agreed SLAs in place with regularly used providers. On the other hand, it may be that in the most dynamic service usage situations optimal, rather than agreed, levels of service are what are sought. This can be achieved through QoS-based discovery alone.

A further issue with SLAs is tampering: even if both parties hold a copy of the SLA it is hard to resolve disputes if one party alters the document after the agreement is made. Therefore, either an infallible SLA validation mechanism or a trusted third party to hold or certidy the SLAs may be required. This could possibly be provided along with the negotiation service.

3.8 QoS Delivery

In order to be able to make meaningful promises in their Service Level Agreements providers must have mechanisms within their service architectures to allow provision of known quality levels. Much work exists which could be applied to QoS provision, but not on applying it in service-centric systems.

At the upper layer of the OSI network stack, SOAP is generic enough that it in no way hinders the implementation of any service level QoS mechanisms. For instance it is easy to formulate a SOAP message to contain a QoS parameter, a QoS query, response, etc. However, since the

majority of communications between clients and services use SOAP it is important that these communications themselves be subject to network QoS constraints. It is possible to enforce such constraints on SOAP's network performance by selecting appropriate QoS mechanisms at lower OSI layers (e.g. RSVP, DIffserv), but there is no standard mechanism for doing so. A hypothetical QoS header block is discussed in the SOAP 1.2 primer, but this suggestion would not be trivial to implement, as the existing HTTP binding would need to be extended. Directly QoS enabling SOAP itself is therefore some way off and it is at the TCP level that network QoS is likely to be enabled. SOAP is not quite ubiquitous in the service world: the main alternative, REST (Representational State Transfer), eliminates the extra protocol layer and uses standard HTTP methods and existing ways to process XML. The question of how to select underlying QoS mechanisms remains the same however.

The related problem of mapping from QoS specifications to resource configuration and allocation using underlying QoS provision mechanisms seems to be almost as important as developing the mechanisms themselves. How to inspect the underlying mechanisms for information on what levels of QoS can be delivered is also important.

3.9 QoS and Service Composition

Composition, particularly dynamic composition (i.e. that done using late binding at runtime), affects all aspects of using QoS in service-centric systems. The composition of QoS requirements, capabilities, measurements and SLAs is potentially very complex. The problem gets more complex as more forms of composition are used, more complex metrics are employed, and as the composition itself becomes more complex.

If the client is creating the composition themselves then they must make sensible decisions regarding how they handle SLAs and negotiation. For instance, if in the 'travel agent' scenario a number of search results from various providers are to be aggregated then the cost of calling each one must be taken into account and the QoS agreements aggregated correctly. For instance, the overall guaranteed response time in this case is the maximum of the guarantees received. On the other hand, another hypothetical application may be seeking to minimise the time taken for an operation and may therefore take the first valid result set (thus the guaranteed response time is the minimum of the guarantees received). Each service will have to be paid for, but the client may find that the improved over all response time is worth the extra cost. The correct aggregation therefore depends upon the form of composition and the nature of the metric involved.

If an intermediary service or middleware component composes services for clients transparently (e.g. in the simplest case, one provider subcontracts another to provide some piece of functionality) then they will have to perform all of these calculations and present the correctly aggregated SLA to the client. A measurement service may provide measurements on composite services by aggregating them in the same way. There is potential for links to BPEL or other composition languages to facilitate all of these compositions of QoS.

4 Existing Work on Quality of Service Specification

4.1 HQML

The Hierarchical QoS Markup Language (HQML) [9], developed at the University of Illinois, is an XML based language for enhancing distributed multimedia applications with QoS capabilities. Hierarchical refers to the fact that there are three levels of specification in HQML: User, Application and System Resource Level.

Its particular aim is to provide tags to allow intelligent/adaptive middleware to improve the provision of QoS automatically through such self-explanatory application layer tags as <ReconfigRule>, <Condition> and <Action>. It also allows feedback to and from the user (e.g. when adaptations occur) primarily through the tags <Notification> and <Feedback>.

Resource level in HQML includes the familiar network performance characteristics as well as host system hardware characteristics such as <CPU>, <Memory> and <Disk>. The application developer should not have to deal directly with these parameters. A visual programming environment aids in the the generation of HQML files and a compiler maps from application to resource level specifications. A special boundary symbol relational (SR) grammar named ConfigG allows formal consistency checking of the developer's visual QoS specification and automatic HQML generation.

Despite its name HQML bares no relation to QML and is designed specifically for use in distributed multimedia applications. Its schema is therefore quite inflexible and does not apply well to wider services. It also lacks any richness of expression beyond bounds on parameters.

One interesting feature the language allows for is the inclusion of different pricing models in the user level specification. The usage of QoS agreements in practice will clearly involve relating quality to cost. It therefore seems that languages for expressing SLAs must address costing as well as QoS specification.

4.2 OWL-S

OWL [7] is the Web Ontology Language: a semantic markup language for publishing and sharing ontologies on the World Wide Web. An ontolgy is defined as "an explicit formal specification of how to represent the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them". OWL-S [11] is an OWL ontology allowing for semantic markup of web services. One of its intended uses is service QoS specification.

In OWL, Classes provide an abstraction mechanism for grouping resources with similar characteristics. The OWL-S ontology is structured around the Service class. One instance of this will exist for each service. It has three properties:

- **presents**, which is a descendent class of ServiceProfile, and describes what the service provides and requires.
- **DescribedBy**, which is a descendant class off ServiceModel, and describes how the service works.
- **supports**, which is a descendant calss of ServiceGrounding, and describes how to use the service.

It is the presents property and its type ServiceProfile which are of interest in specifying the QoS a service provides.

The basic Profile class in OWL-S consists of some contact information and a functional description (preconditions, inputs, outputs, effects). On top of this there are a service category, a quality rating and an unbounded list of parameters. The categorisation depends upon the provider using some appropriate categorisation scheme (i.e. OWL-S alone does not solve the "I want a service of type X" problem). The same is also true of the quality rating. These are therefore of limited use. The final parameter list remains entirely proprietary - but could be used for QoS data in the absence of any standard extension to the ontology for QoS.

Although this base class Profile can be used directly, for QoS purposes it is more likely that OWL subclassing would be utilised to provide a more specific service representation with enhanced QoS representation capabilities. DAML-QoS [1] is an attempt to provide a QoS ontology to complement OWL-S (or DAML-S as it was called at the time DAML-QoS was created). This is based around a QoSProfile class.

A great advantage of using OWL-S for the purposes of QoS description would be that QoS would then fit into the wider semantic service description (i.e. it would be expressed using the same ontology and could related to semantic as well as syntactic elements of the service description). OWL-S also has a number of supporting tools and APIs. A possible downside to using OWL-S for QoS purposes is that may not wish to specify in detail the functional aspects of your service. In this case the DescribedBy property might be left empty.

4.3 QoS Modelling Language (QML)

QML [10] was developed by Hewlett Packard as a general-purpose means of describing the QoS properties of software components. In QML the emphasis is placed upon QoS specification; the means of reaching such a specification (e.g. negotiation) are not dealt with.

The basic element in a QML specification is known as a contract. The contract is bound to a software interface, operation, operation argument or operation result using the language element known as a profile. Each contract is of some specified contract type. The contract type specifies the dimensions that can be used to specify QoS properties within some category (e.g performance, availability, security, timing). This is done by stating a domain of values which the dimension may take. The domain may be specified numerically, as a set or as an enumeration. QML is not prescriptive as to what dimensions may be used; the vocabulary of the language may be extended with user-defined dimensions. A contract is made up of constraints on the dimensions of its contract type, or more often as constraints on aspects of dimensions. The built-in aspects are percentile, mean, variance and frequency - but again these can be extended by the user.

A property which may be expressed quantitatively is specified using a dimension with a numerical domain. The ordering of levels of quality comes naturally in such cases; although whether higher values or lower values are better must be stated using the increasing or decreasing keywords. It may be necessary to express an ordering in qualitative properties as well however. For enumerated domains an order is specified on the named elements of the enumeration and for set domains the same technique is used in conjunction with subset inclusion. Properties with ordered domains are subject to "stronger than" and "weaker than" comparisons, allowing richer conformance checks than simple equality. It also allows for a simple notation for creating contracts which refine existing contracts

Despite its relative age QML is generic enough to apply reasonably well to modern service architectures. It does not concentrate on network performance, nor is it limited to numerical parameters. However (admittedly in common with the majority of other languages) the whole area of how to reach a QML contract is left undefined. Whilst specification and negotiation are logically separate it would simplify the QoS usage scenario to design a negotiation language or protocol with reference to a specification language (and vice versa).

Even as a pure specification language QML has its shortcomings. There is no way to define relationships between contracts for particular QoS categories. This applies equally to specifying trade offs between dimensions within any given contract. The result of this is that there is no way to express interdependencies such as that in the 'travel agent' scenario where the provider will only guarantee the time to complete when server loads are not excessive. The logical layer of language, which it was noted above was required to allow more complex expressions such as these, is absent. QML also lacks in the area of expressing what actions should occur if QoS requirements cannot be met at runtime.

In terms of use in service-centric systems, the fact that QML is not XML-based is a major problem. The standards and tools used in service architectures for manipulating XML could not be used, and the method of linking QoS guarantees to the functional aspects described in WSDL would have to be cumbersome.

4.4 QuO/QDL

The Quality Objects (QuO) framework [6] and its CORBA-based implementation support the development of distributed applications with QoS requirements. The framework includes a suite of QoS Description Languages (QDL) designed to enable various aspects of developing QoS-enabled applications. These include a Contract Description Language, a Configuration Setup Language and a Structure Description Language. Where certain QoS properties are not fully facilitated by these languages additional property specific languages exist. Code generators weave code generated from QDL specifications into the target QuO application.

A contract in QuO includes the desired level of service, the level of service it is considered possible to provide and actions to take when the level of QoS changes. QuO and QDL fail in the

same place that many other languages do: The type of requirements it is possible to specify are not rich enough and it does seem rather tied down to CORBA. It is also a non XML language.

However, one great advantage of QuO and QDL is that there is a concrete link between requirements and provision, with integration of existing work such as AQuA for dependability and TAO for real-time support allowing the provision of various QoS properties. This highlights the lack of any such concrete link in the majority of other languages. Whilst QoS specification is a logically separate step to provision, it does sometimes seem meaningless to be able to specify QoS requirements if nobody knows what mechanisms are available to meet them and therefore what agreements it is even possible to make. This is, admittedly, not necessarily the job of the designer of a QoS specification language, but in relatively youthful architectures (e.g OGSA) this gap can pose a real problem.

4.5 SLAng

SLAng [2] is an XML-based language for describing SLAs, which concentrates on component based systems (particularly J2EE and CORBA). Like WSLA, its origin is in the world of web services and E-business. It therefore closely matches the understanding of the term QoS expressed in this paper. SLAng provides a format for QoS negotiation and contract specification and is also designed to be appropriate as input to automated reasoning systems or QoS-aware adaptive middleware.

A key element of SLAng is its aim to facilitate different levels of QoS abstraction (which relates well to the description of service-centric QoS given earlier). At the network layer it is based on Service Level Specification (SLS) Proposal from the TEQUILA project (as submitted to the IETF). The other layers it identifies are the (QoS-aware) middleware layer and the application layer. SLAng also identifies the need for different levels of expressiveness for Horizontal and Vertical SLAs: Horizontal referring to contracts between coordinated peers (e.g. in order to provide replication of a service); Vertical referring to contracts between parties and their underlying infrastructure (e.g. their ISP).

The structure of an SLA in SLAng consists at the top level of an indication of whether it is horizontal or vertical. Below this comes the SLA type, which is one of seven set types including, for instance, application (a vertical SLA type) and networking (horizontal). The SLA then contains responsibilities in three sections: client, server and mutual. In each of these sections, parameters specific to the SLA type (and therefore limited to those defined in the language itself) are defined. The structure of a SLAng SLA is therefore as shown in Figure 1.

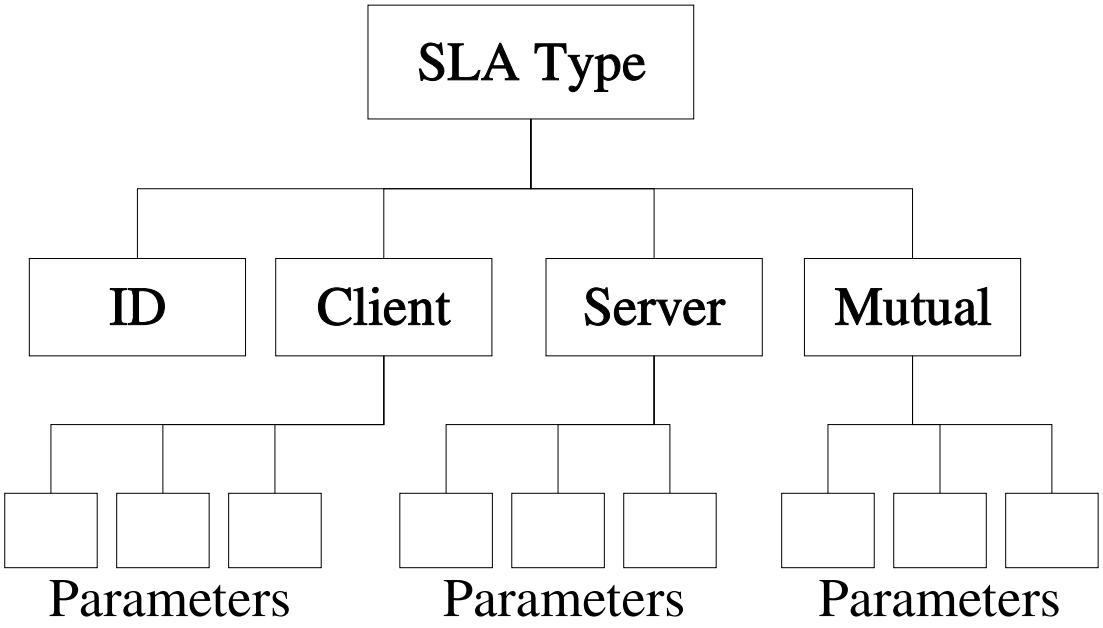
4.6 WSLA

The Web Service Level Agreement (WSLA) language [5], developed by IBM, is, as the name suggests, a specification language for Service Level Agreements (SLAs) for Web Services. It is based upon XML and aims to be applicable to any domain. One of the main ways it achieves this is by allowing domain-specific language extensions. The mechanism for doing this is type derivation by XML schema. A set of standard extensions also exists to cover common usage situations. As well as the language itself, a WSLA framework has also been defined.

```
<wsla:sla xmlns:xsi="http://www3.org/2001/XMLSchema-instance"
xmlns:wsla="http://www.ibm.com/wsla"
name="StockquoteServiceLevelAgreement12345">
  <parties> ... </parties>
  <servicedefinition> ... </servicedefinition>
  <obligations> ... </obligations>
</wsla:sla>
```

As shown above, the first section of an agreement in WSLA specifies the parties involved. As well as signatory parties (the service provider and the client) WSLA allows supporting parties to be specified. One particularly useful result of this is that a trusted third party for measurement and

Figure 1: Structure of a SLAng SLA



monitoring may be agreed. The service definition section follows. For each QoS parameter a metric is defined and for each metric a Measurement Directive is specified. This directive contains the information needed to retrieve the metric. Composite metrics (e.g. average, minimum, maximum, etc.) may be specified as a Function in the SLA or exposed by the service provider as a well-defined interface for further processing. SLA Parameters map to a metric and may be associated with a permitted range for a specific customer. The final section of the SLA is the parties' obligations. This may contain two types of obligation: the Service Level Objective and the Action Guarantee. The first type guarantees a particular state of SLA parameter in a given time period. The second guarantees that a specified action will be performed in given circumstances. This is particularly useful in specifying the desired behaviour when a service fails to comply with some element of the SLA. Uniquely among the languages discussed here, WSLA allows logical expressions in its obligations section. For instance the example expression below would apply in the 'travel agent' scenario, allowing some freedom in the guaranteed response time of the booking service under heavy server loads. It states that response time must be less than 0.5 unless the transaction rate is greater than 10000.

```
<Expression>
 <Or>
   <Expression>
     <Predicate xsi:type="Less">
       <SLAParameter>ResponseTimeThroughputRatio</SLAParameter>
       <Value>0.5</Value>
     </Predicate>
   </Expression>
   <Expression>
     <Predicate xsi:type="Greater">
       <SLAParameter>TransactionRate</SLAParameter>
       <Value>10000</Value>
     </Predicate>
   </Expression>
 </Or>
```

```
</Expression>
```

A number of ways of establishing a SLA are contemplated within the WSLA framework: ranging from the service provider offering one fixed SLA through the provider offering a service with differing (but fixed) SLAs, to a SLA being fully negotiated between the parties. The details of negotiation are not specified in WSLA, but ebCPP [3] is hinted at as a possible means of doing so.

Importantly, WSLA is thoroughly documented, a full language specification is publicly available, and it exists as part of a wider framework. This openness and completeness can only aid uptake of the language and make its application much clearer than some other QoS languages.

WSLA is also extensible, XML-based, allows logical expressions such as trade-offs between parameters to be expressed and allows failure behaviour to be specified. All of this makes WSLA seem very promising as a QoS/SLA specification language; its use in practice being the only missing piece of the jigsaw.

4.7 WS-Policy

WS-Policy [8]provides a grammar for defining capabilities, requirements and characteristics of web services. QoS is one of the things which can potentially be expressed using WS-Policy. WS-PolicyAttachment defines mechanisms for associating such a policy with elements of XML, WSDL and UDDI.

The normal form of policy expression (which should be used where possible) is as shown below:

```
<wsp:Policy ...>
  <wsp:ExactlyOne>
    <wsp:All> [ <wsp:Assertion ...> ... </wsp:Assertion>]* </wsp:All> ]*
  </wsp:ExactlyOne>
</wsp:Policy>
```

The ExactlyOne tag contains policy alternatives. Each alternative consists of a number of policy assertions (contained within the All tag).

WS-Policy is lightweight: providing no advantages for QoS specification other than that it is a standard way of associating QoS-like descriptions with a service. It is therefore only of real interest in specifying service QoS capabilities. Here, a WS-QoSPolicy could be standardised (as happened with WS-SecurityPolicy[4]). This would mainly involve specifying Assertions relating to QoS. This might be best realised by defining bindings to one or more of the QoS Specification Languages discussed here.

4.8 Other QoS Specification Languages

- XQoS is an XML based language for QoS specification developed at Ensica's Departement Mathematiques appliquees et Informatique. XQoS applies mainly to multimedia systems: one of its main observations being the need for both intra and inter-flow QoS specifications. The basis of the language is the Time Stream Petri Network (TSPN) model. This model is particularly suitable for modelling synchronization issues in concurrent streams or processes. The concentration on stream-based QoS means that this type of language does not apply to service-oriented architectures well and is typical of distributed multimedia specific languages.

- The Quality Assurance Language (QuAL) is similar to XQoS in that it is based upon the Time Stream Petri Network (TSPN) formal model. It facilitates optimal QoS mapping from application level requirements to underlying communications service specifications.

- Service Level Specifications (SLS) are being standardised as part of the TEQUILA project. In this context an SLS refers to network QoS in a public IP network and applies to a unidirectional traffic flow.

- Work is also under way on a UML Profile for specifying QoS with a number of proposals having been put forward.

5 Conclusion

How QoS should be best employed in service-centric systems depends to some extent on what usage models are widely adopted. The level of adoption of service discovery and SLA negotiation mechanisms are central to this. Resistance to the uptake of these technologies is likely, in part because people find them harder to trust than existing human equivalents.

If the most dynamic usage scenario is to come about, a QoS and SLA specification language will be one of its main technological foundations. Research on QoS-based discovery, SLA negotiation and therefore measurement and monitoring all depends to some extent on a successful QoS/SLA specification language. Based upon the discussion in Specification Language section and the survey of existing languages it is possible to make the following observations about the requirements of a service-centric QoS specification language:

- **It must be extensible**. The factors of interest may vary widely between services. A usable QoS specification language is therefore likely to be modular and extensible. It should not necessarily attempt to define an exhaustive list of parameters, metrics, etc. but allow domain specific extensions to allow for completeness.
- **It should be XML based**. This would allow integration with the XML based service architectures (including existing languages such as WSDL, BPEL) and holds advantages such as standards and tool support for verification, translation and searching.
- **It must allow more complex specifications than simple bounds**. In particular it is often the case that one parameter may be traded off for another or that it is acceptable for levels of service to degrade in certain aspects/certain ways. In order to encompass these and other possibilities it is proposed that a logical layer of language beyond the purely declarative is needed.
- **It must include failure and non-compliance semantics**. It is important not only to be able to make agreements about what happens under normal conditions - but what happens in exceptional conditions. If it is important that certain QoS agreements are met then the actions to take when they cannot or are not met are equally important.
- **It cannot exist in isolation**. The QoS specification must exist within some environment in order to hold any meaning. Therefore, as noted above, how QoS is applied in practice is a key element in deciding what a useful language would consist of. The language must therefore reference factors such as measurement, prediction, monitoring, compliance checking, provision and negotiation and should integrate with standards and languages in these areas.

WSLA already goes some way to meeting all of these criteria. However, it may be over-engineered, and it remains to be seen whether it gains widespread support. On the other hand, OWL-S needs further extension to fully meet the requirements - but given its wider remit of semantic description it is perhaps a good candidate for the role of QoS description too. Such an extension of the OWL-S ontology should be informed by the specification languages discussed here. Perhaps the most obvious way of all to publish QoS capabilities is by extending WSDL. However, separating the functional interface description from the non-functional is important as the functional interface may be specfied by some third party (e.g. a standards organisation) - but non-functional properties will still vary between implementations. WS-Policy is therefore an appealing standard, although it does not solve the QoS specification problem on its own.

Research into QoS-based service discovery is largely entangled with that of QoS requirements specification: the first problem being discovering what the best forms are for QoS requirements and queries (which are essentially of the same form).

Automated SLA Negotiation is an area wide open for research (whether it turns out to be widely used is another question). The process itself must be researched and fine tuned (and the relevant inputs from both parties decided upon). The QoS specification language will describe parts of what are being negotiated, other elements of the full SLA specification language (such as costs, third parties, etc.) will make up the rest. Interaction between the negotiating intermediary and the measurement and compliance monitoring services is also vital to provide meaningful agreements which must be adhered to.

The task of measuring QoS is perhaps the most difficult area open for research. The potential variety of QoS parameters only makes it more difficult. A generic means for specifying metrics and their context is therefore needed. This could then be extended and utilised in domain-specific ways as the need arose. Research into minimising and reporting the effects of measurement and how to compose measurements for composed services also form potentially large research areas.

Research into QoS delivery should address the problem of mapping requirements onto resource configuration and underlying QoS provision technologies. Extending the range of QoS parameters covered by such technologies is also vital, although is likely to be incremental as services are used in more and more domains.

Looking at the whole QoS picture for service-centric systems engineering it is clear that throughout the process there is overlap in the entities referred to at various stages. To enable smooth interfacing between these stages, the most important thing is that the languages and protocols used are standardised. To ensure common terminology and an understanding that applies across the QoS technologies this standardisation should perhaps make use of, and contribute to, a service ontology such as that defined by OWL-S.

6 References

References

[1] Chen Zhou et al. DAML-QoS Ontology for Web Services. http://www.ntu.edu.sg/home5/PG04878518/Articles/icws04_235_Chen_Z.pdf.

[2] D. Lamanna et al. SLAng: A Language for Defining Service Level Agreements. http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/FTDCS03/.

[3] David Burdett et al. Collaboration-Protocol Profile and Agreement Specification. http://www.ebxml.org/specs/ebCCP.pdf.

[4] Giovanni Della-Libera et al. WS-SecurityPolicy. http://www-106.ibm.com/developerworks/library/ws-secpol/.

[5] Heiko Ludwig et al. Web Service Level Agreements (WSLA) Project. http://www.research.ibm.com/wsla/.

[6] Partha Pal et al. Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration. http://csdl.computer.org/comp/proceedings/isorc/2000/0607/00/06070310ab%s.htm.

[7] Sean Bechhofer et al. OWL web ontology language reference. http://www.w3.org/TR/owl-ref/.

[8] Siddharth Bajaj et al. WS-Policy. http://www-106.ibm.com/developerworks/library/specification/ws-polfram/%.

[9] Xiaohui Gu et al. An XML-based Quality of Service Enabling Language for the Web. http://cairo.cs.uiuc.edu/publications/paper-files/jvlc.pdf.

[10] Svend Frolund and Jari Koisten. QML: A Language for Quality of Service Specification. http://www.hpl.hp.com/techreports/98/HPL-98-10.html.

[11] Mark Burstein et al. OWL-S. http://www.daml.org/services/owl-s/.