Requirements on Quality Specification Posed by Service Orientation

Markus Garschhammer and Harald Roelle

Munich Network Management Team
University of Munich
Oettingenstr. 67
D-80538 Munich, Germany
{markus.garschammer|harald.roelle}@ifi.lmu.de

Abstract. As service orientation is gaining more and more momentum, the need for common concepts regarding Quality of Service (QoS) and its specification emerges. In recent years numerous approaches to specifying QoS were developed for special subjects like multimedia applications or middleware for distributed systems. However, a survey of existing approaches regarding their contribution to service oriented QoS specification is still missing.
In this paper we present a strictly service oriented, comprehensible classification scheme for QoS specification languages. The scheme is based on the MNM Service Model and the newly introduced LAL–brick which aggregates the dimensions **L**ife cycle, **A**spect and **L**ayer of a QoS specification. Using the terminology of the MNM Service Model and the graphical notation of the LAL–brick we are able to classify existing approaches to QoS specification. Furthermore we derive requirements for future specification concepts applicable in service oriented environments.

Keywords

QoS specification, service orientation, classification scheme

1 Introduction

In recent years Telco and IT industries have been shifting their business from monolithic realizations to the composition of products by outsourcing, which results in creating business critical value chains. This trend has had its impact on IT management and paved the way for concepts subsumed under the term service (oriented) management. Now, that relations involved in providing a service are crossing organizational boundaries, unambiguous specifications of interfaces are more important than ever before. In federated environments they are a fundament for rapid and successful negotiation as well as for smooth operation. Here, not only functional aspects have to be addressed, but quality is also an important issue.

In the context of service management, the technical term Quality of Service (QoS) is now perceived in its original sense. Prior to the era of service orientation, the term QoS was mainly referred to as some more or less well defined technical criterion on the network layer. Nowadays, QoS is regaining its original meaning of describing a service's quality in terms which are intrinsic to the service itself. Furthermore, QoS now reflects the demand for customer orientation, as QoS should be expressed in a way that customers understand, and not in the way a provider's implementation dictates it.

In the past, a number of QoS specification concepts and languages have been proposed. Unfortunately, when applied to real world scenarios in a service oriented way, each shows weaknesses in different situations. For example, considering negotiations between a customer and a provider, some are well suited regarding customer orientation, as they are easily understood by the customer. But they are of only limited use for the provider's feasibility and implementation concerns. Other specification techniques suffer from the inverse problem.

Apparently there is room for improvement in service oriented specification of service quality. This paper contributes to the field by introducing a classification scheme for quality specification techniques which is strictly service oriented. This is accomplished by considering e.g. the service life cycle, different roles and types of functionality. By applying the classification scheme to representative examples of quality specification techniques, the current status in the field is outlined. To contribute to the advancement of service orientation, we derive requirements for next generation specification techniques by two ways: First we analyze today's works' flaws and second we deduce requirements from our classification scheme.

The paper is organized as follows. In the next section (Sec. 2) the classification scheme for QoS specification languages and concepts is introduced. Application to typical examples of QoS specification techniques is discussed in Sec. 3. Using these results, the following section (Sec. 4) identifies requirements for future quality specification approaches. Section 5 concludes the paper and gives an outlook on further work.

2 Classification Scheme

In this section a classification scheme for quality specification concepts and languages is developed. In doing so, the paradigm of service orientation is strictly followed. Multiple aspects of services are covered, functional aspects of a specification as well as its expressiveness in relation to service properties.

In order to develop the classification, the MNM Service Model [GHH+02,GHK+01] is used as the foundation for the classification. It is a bottom–up developed model which defines a common terminology in generic service management, specifies atomic roles and denotes the major building blocks a service is composed of. Doing so, it offers a generic view on the building blocks rather than a specification for direct implementation. As the MNM Service Model is a generic model, which is not focusing a certain scenario, it serves well as a starting point for our classification, in respect to develop a model where completeness, generic applicability and scenario independency is ensured.

The second ingredient for our classification is the set of common concepts that can be found in various quality specification schemes. This set was derived from the survey of Jin and Nahrstedt [JN04] and is an enhancement of the taxonomy presented there.

2.1 The MNM Service Model as a Map for Specification Concepts

The MNM Service Model offers two major views (Service and Realization View) which group a service's building blocks into different domains, according to their roles and related responsibilities. Figure 1 combines the two views. One major characteristic of the model is the so called *Side Independent Part*[1]. Beside the *service* itself, it depicts additional building blocks which should be specified independently from realization details on either the *provider side* and the *customer side*.

The MNM Service Model decomposes the specification of a service's quality in two parts. The first part describes quality relevant properties of a *service* (class *QoS Parameters* in Fig. 1). The second part poses constraints on those specified properties which have to be met by the provider and are agreed upon in an *service agreement*. For both parts, relevant properties of the system have to be defined in an unambiguous manner.

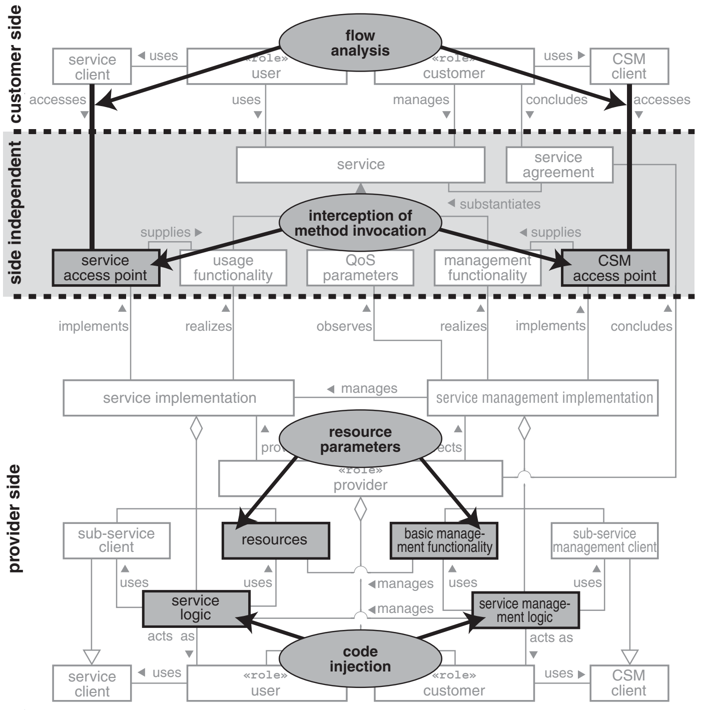

Fig. 1. Reference Points located in combined view of the MNM Service Model

QoS parameters may be specified against different reference points. Thus, even when they bear the same name, they may have different semantics. For example, a delay

[1] In the following, parts of the model will be printed in italics

specification could be measured at the *user's client* or inside the *service implementation*, which results in different QoS parameters. In fact, our extension of the taxonomy presented in [JN04] describes such possible reference points for quality specification.

By locating the reference points in the MNM Service Model characteristics of these reference points can be identified. First of all, the model's part where the reference point is located, enables us to identify the affected roles. Furthermore, this allows us to draw conclusions on dependencies to other parts of a service. Thus we can identify typical properties of reference points, like limitations regarding portability to different realizations or the applicability in situations, when service chains come into play.

The following paragraphs first describe those reference points. In each paragraph's second part the reference point is located in the MNM Service Model as depicted in Fig. 1. By this, basic characteristics of specification techniques using the respective reference point can be pointed out later on by simply marking the corresponding reference point in the MNM Service Model.

Flow/Communication Most of today's common QoS parameters, such as throughput or delay, are measured and thus specified from a communications point of view. Quality related properties of a service are derived from properties of a data stream or, in general, a flow. Constraints on the quality of a service are simply mapped onto constraints of the flow (e.g. "the transmission delay must not exceed 10ms"). So the quality of a service is only implicitly defined by properties of the communication it induces. This definition is therefore at the risk of being too coarse in respect to the service's functionality. However, this way of expressing quality is widespread because properties of a flow can be easily derived in real world scenarios. A typical example would be an ATM based video conferencing service where its properties are described as ATM QoS parameters.

In the MNM Service Model a communication flow in general can be observed between a client and the corresponding service access point (SAP). This relation exists between the *service client* and *service access point* (when accessing the service's *usage functionality*) as well as between the *customer service management (CSM) client* and the *customer service management (CSM) access point* (when accessing the *management functionality*). Hence, a quality specification has to be applied not only to the usage functionality but also to the management side.

As can be seen in Fig. 1, the relation between the *service client* and the *service access point* crosses the boundary between the *customer side* and the *side independent* part of the model (the same applies for the management side). Any analysis of flows depend on the service clients, thus, it cannot be implementation independent. In consequence, specifications using the technique of flow analysis depend on a client's implementation as well.

Method/API Invocation Another technique to derive quality relevant properties of a service is motivated by object oriented (OO) design, programming and middleware architectures. Here, quality is specified as properties of a method invocation, e.g. the time it takes a method for encoding a video frame to finish. Constraints on these properties can be posed as easily as in the former case. This method of quality measurement and description requires the interception of method invocations. As this is naturally done in component oriented middleware, this technique is mostly used there.

Method invocation may occur at almost any functional component of the MNM Service Model. However, the invocation interception concept used in OO environments or middlewares can be mapped best to the service's access points where methods in the sense of service (management) functions can be invoked. The idea of interception of method invocations is therefore depicted in Fig. 1 at the *service access point* and the *customer service management (CSM) access point*. As this concept only uses blocks of the model which are located in the *side independent* part, it does not depend on any implementation, neither on the *customer* nor on the *provider side*.

Code Injection The idea of code injection is to directly integrate constraints on quality into a service's implementation — into its executable code. Steps of execution monitored by injected code yield service properties (such as processing time or memory consumption). Constraints on these properties are inferred by directly coding conditional actions to be executed when a constraint is satisfied or violated. For example, information on memory usage during the decoding of a video stream is measured. If it exceeds a certain value, a less memory consuming but also worse performing decoding method is used. This procedure automatically assures a certain quality, in this case a guaranteed maximum amount of memory used.

The MNM Service Model divides a service's implementation into three parts: *subservice client*, *service logic* and *resources*. The *service logic* orchestrates resources and subservices to implement the service's *usage functionality* as a whole. The idea of code injection, in the sense of the service model, is to enhance the *service logic* with inserted code to automatically derive properties of the running service. Observation of these properties and reaction to changes are directly coupled. As the Service Model distinguishes between a service's *usage functionality* and its *management functionality*, this concept is shown in both the *service logic* and the *service management logic*. As one can easily see, the idea of code injection depends directly on a service's implementation by a provider. It is therefore an instrument for providers to assure a certain quality, but obviously should not be used in service negotiation when customer and provider need to establish a common understanding of a service's quality parameters.

Resource Parameters Quality relevant properties of a service can also be derived from the parameters of resources the service is realized with. For this purpose, resource parameters can be aggregated in various ways.

However, details of the gathering and aggregation process have to be defined after service deployment, because relevant details of concrete resources used are unknown before deployment. Even worse, the specification may have to be adapted on a service instance basis because different service instances might use different resources, whose concrete characteristics might be needed for specification. Constraints on these resource oriented properties can be posed at various aggregation levels, but their derivation from constraints posed on the service itself is not a trivial task.

In the MNM Service Model, information about resources can be directly gathered from the *resources*, but can also be obtained via the class *basic management function-*

ality. When specifying quality aspects of the *management functionality* the *basic management functionality* itself is targeted in a QoS specification. As the location of both *resources* and *basic management functionality* inside the provider's domain illustrates, even with a suitable aggregation method, this concept of specification can only express quality that directly depends on the provider's own implementation. As most services realized today depend on subservices, this specification can be used for basic services only.

By introducing the various reference points, locating them in the MNM Service Model and by identifying their basic properties and limitations, the first part of our classification scheme is now explained. While up to here our analysis focused mostly on functional aspects of the MNM Service Model's views, additional non–functional aspects have to be regarded for a comprehensive classification of quality specification techniques. This will be carried out in the next section.

2.2 Dimensions Covered by Quality Specifications – the LAL-Brick

Apart from its decompositions into functional building blocks and related roles, as described with the MNM Service Model, a service can also be described in another view, which focuses on non–functional properties of a service. As shown in the following paragraphs, we describe this non-functional view using a three dimensional brick, depicted in Fig. 2. The brick's three dimensions are ***L**ife cycle*, ***A**spect*, ***L**ayer* and is therefore called the LAL–brick from here on. The axes of the brick, its dimensions, can be marked with typical properties. A tiny cube is attached to each property and as the dimensions are independent from each other, all tiny cubes together form the brick. The dimensions and their properties are described in the following.

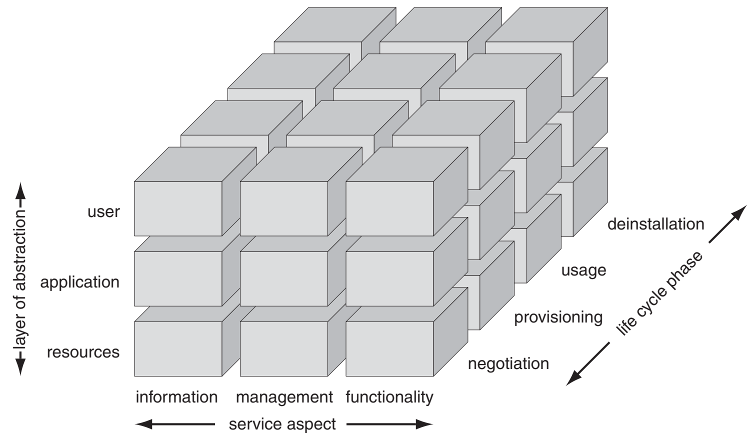

Fig. 2. Dimensions of quality specification arranged in a brick

Approaches to specify QoS can easily be depicted in the brick by simply marking the cubes corresponding to the properties this specification approach fulfills. Different "marking patterns" of different approaches explicitely visualize their classification.

Life Cycle The process traversed by a service – when seen as an object of management – is called the life cycle. This process can be split up into different phases. According to [GHH$^+$02] it begins with the *negotiation phase* where *customer* and *provider* determine a service's functionality, its QoS and suitable accounting schemes. In the next phase, the *provisioning phase*, the provider implements the service using its own resources and subservices he buys (then acting in the role of a *customer*). When the implementation is completed, the *usage phase* begins with *users* actually using the service. The life cycle ends with the *deinstallation* of the service.

The mapping of the reference points on the MNM Service Model already suggested that the service life cycle is a relevant dimension in quality specification. For example, as explained above, specification schemes based on *resource parameters* have a strong relation to the finalization of the *provisioning phase*, while using *method/API invocation* as reference points, quality specification could be fully done in the *negotiation phase*. Thus, concepts and methodologies dealing with service should be aware of the life cycle and ensure reusability of results they deliver to other phases. At least they should explicitly denote which phase they cover or were designed for.

Aspects of a Service The notion of a service not only defines a set of functions accessible to a *user*. In an integrated view it also defines how the management of a service is accomplished by the *customer* (see Fig. 1). As shown above, independently from the type of reference points used in a specification mechanism, *management functionality* must be targeted as well as a service's *usage functionality*. Of course, when a service is specified, the content it delivers or the information it deals with are defined. In consequence, this information might be subject to quality considerations as well. From now on, we use the notion of aspects of a service to denote the triple of *function*, *management* and *information*.

Layer of abstraction When concepts or methodologies dealing with services are presented, different layers of abstraction can be recognized. Some ideas focus on the *resources* that a service is built upon, some describe a service from an *application's* point of view. At last, the service can be described from an *user's* point of view (as of [JN04]).

Service orientation demands concepts spanning all three layers of abstraction denoted above, so providers, customers and users can use them. At least, mappings between the different layers should exist so that an integrated concept could be built up out of ideas only spanning one layer of abstraction.

2.3 Comprehensible, Service Oriented Classification Scheme

The set of reference points marked within the MNM Service Model in conjunction with the LAL–brick now delivers a comprehensive classification scheme for approaches specifying QoS. It should be emphasized that reference points are not exclusive to each other. This means, that a concrete quality specification mechanism might use several types of reference points. Section 3.4 shows an example for this.

The MNM Service Model and the LAL–brick offer different views on the specification of QoS. The Service Model, used as a map to visualize different reference points, focuses on functional aspects, whereas the LAL–brick gives an easy to use scheme to denote non-functional properties of specification techniques. Together, both views offer the possibility of a comprehensive classification of existing approaches in QoS specification, as will be shown in the following section.

3 The Classification Scheme Applied – State of the Art in QoS Specification by Example

After presenting a comprehensive and service oriented classification scheme in the previous section, we will now discuss typical representatives of specification languages. Each specification language realizes one of the approaches denoted in Sec. 2. We do not give a full survey on QoS specification languages and techniques here. But we demonstrate the application of our classification scheme to existing approaches in order to derive requirements on a service oriented QoS specification in the following Sec. 4.

3.1 QUAL – a Calculation Language

In her professorial dissertation [DR02a] Dreo introduces QUAL as part of "a Framework for IT Service Management". The approach of QUAL as such is also presented in [DR02b]. The key concept of QUAL is to aggregate quality parameters of devices (named as quality of device, QoD) to basic quality parameters which themselves can be aggregated to service relevant QoS parameters. The aggregation process is based upon dependency graphs which describe service and device interdependencies.

As QoD is gathered on the resource level, QUAL obviously uses *resource parameters*. Although QUAL can express higher level quality parameters at the application level, they always depend on the on the QoD gathered from the resources. Thus, *resource parameters* are the only reference point directly used in QUAL, as application level QoS is specified through aggregation.

QUAL covers a wide range of abstraction from resource to service oriented quality parameters. However, QUAL does not directly address the specification of user–oriented QoS. In our classification scheme, it therefore covers the two lower abstraction layers, *resource layer* and *application layer*. QUAL focuses on the *functionality aspect*, the *management aspect* and the *information aspect* are not explicitly mentioned.

As QUAL is based on resource parameters, its application is restricted to the *usage phase* of the life cycle where these parameters are available. Even though QUAL covers only the usage phase, it is highly dependent on specifications and decisions made in the *negotiation* and *provisioning phase*. This results from the fact, that aggregation of quality parameters is based on dependency graphs which have to be determined before QUAL is applied.

3.2 QDL – QoS Description Language

QDL [PJS$^+$00] is an extension to the interface description language (IDL) [ITU97] which is used to specify functional interfaces in CORBA [COR04]. It is the description language used in the QuO (Quality of Service for CORBA Objects) framework introduced in [ZBS97]. The key concept of QuO is to enhance the CORBA middleware concepts with QoS mechanisms. For this purpose, additional facilities are added to the CORBA runtime to ensure a specified quality. The desired quality is determined in QDL and its sublanguages.

Based on QDL statements, extra code is generated in the QuO framework which is joined with the functional code when the corresponding object files are linked together. Thus, QDL uses *code injection* as a reference point for the specification of QoS. By using CORBA as an implementation basis, QuO and QDL abstract from real resources and specify QoS at the *application layer* of abstraction. Naturally the CORBA based approach limits the expressiveness of QDL and prevents the specification of user–level QoS.

QDL only covers the of *aspect of functionality* and does not mention any possibilities to extend its approach to the other aspects *management* and *information*. QDL, together with the supporting framework QuO, covers the life cycle phases *provisioning* and *usage*. The reason for this is, that code executed in the usage phase is automatically generated from specifications laid down in the provisioning phase, when a service is realized according to customer's needs.

3.3 QML – Quality Modeling Language

The Quality Modeling Language QML [FK98] was developed at HP-Labs, another, quite similar approach was presented in a thesis [Aag01]. QML separates the specification of (desired) quality from its assurance or implementation respectively. As specifications are bound to abstract method definitions of the considered application, QML uses *method invocation* as the reference point. The authors of QML also propose a corresponding runtime engine that could be used to implement the specifications made in QML.

Thus, the system as a whole (QML and its runtime engine) offers support for the whole service life cycle: As specifications made in QML are independent of an implementation, they could be easily used in the *negotiation phase*. *Provisioning* and *usage phase* are supported by the runtime engine QRR (QoS Runtime Representation) which unfortunately has not been implemented yet.

Obviously, due to their binding to abstract methods, specifications in QML are made at the *application level* of abstraction. As long as resources are encapsulated in an (object oriented) interface, QML specifications might be used at the *resource level* as well. However, this possible extension is not mentioned by the authors of QML. A distinction of different aspects of QoS is not made either. QML, like all the other specification languages introduced so far, definitely focuses on the aspect of *functionality*.

3.4 QUAL – Quality Assurance Language

The quality assurance language was introduced in [Flo96] as part of QoSME the QoS Management Environment. Although equal in names, QUAL by Florissi and QUAL introduced at the very beginning of Sec. 3 follow quite different approaches. QoSME–QUAL specifies quality in relation to communication properties observable at a so called port. So, it uses a *flow of communication* as reference point. QoSME also provides a runtime engine to ensure the specifications made in QUAL. As this engine is directly woven into an application's executable code, QoSME uses the concept of *code injection* as well. Even though this is only done for the assurance and not for the specification of QoS, it leads to a form of specification closely related to the executable code.

QUAL statements are not very meaningful without a specific implementation in mind. Thus, QUAL cannot be used during the *negotiation* of a service, where an implementation is not yet existent. However, QUAL supports the *provisioning phase* by QoS specification directly attached to the code to be executed later. Together with the runtime system of QoSME, QUAL also supports the *usage phase*.

QUAL claims to specify QoS at the *application layer* of abstraction but does neither mention nor address the other abstraction layers (resources and user). Because QUAL analyzes communication flows, it primarily covers the *aspect of functionality*, but could in some sense also be related to the *aspect of information* when the content of flows is examined.

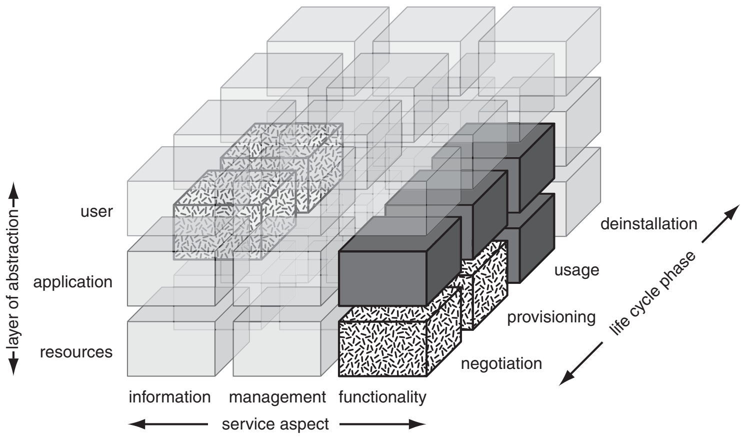

Fig. 3. Approaches marked in the LAL-brick

3.5 The Big Picture

To conclude our presentation of existing approaches we again show the LAL–brick in Fig. 3. In this figure, the parts covered by the reviewed specification languages are marked. Possible extensions of existing approaches, as mentioned above, are spotted, whereas the parts exactly matched are marked dark grey.

As one can easily observe, huge parts of the LAL–brick are not covered at all. Requirements for future specification languages resulting from this “gap” are discussed in the next section.

4 Directions and requirements for future research

As the previous section shows, current work has deficiencies regarding extensive service orientation capabilities. By summing up the flaws and comparing it to the classification scheme, this identifies new requirements which should be met by the next steps in the field. The classification scheme of Sec. 2, which consists of the set of reference points on the one hand, and of the LAL–brick on the other hand, is used here again.

As a service oriented, generic and comprehensive solution for QoS specification technique is required, a full coverage of the LAL–brick should be achieved. Therefore, the still missing cubes in the LAL–brick are investigated. Additionally, in conjunction with the MNM Service Model as generic reference, the individual limitations of the reference points, induce additional requirements.

Side independency Following the MNM Service Model, QoS should be described in a side independent manner. As already explained in Sec. 2.1, side dependency is influenced by the actual set of reference point used. A number of specification schemes suffer from the problem, that they exclusively focus on reference points which import realization dependency by design. Namely, just using *Code Injection* or *Resource Parameters* induces dependencies on the provider’s realization of a service. In case of *Code Injection*, using a middleware architecture mitigates the dependencies from specific resources, but being specific on this middleware persists. Relying solely on *Resource Parameters* is even more problematic as only provider–internal resources, but not the subservices purchased by the provider are reflected. Additionally, side independency is not only desirable in the relation to customers. With quality specification being driven by a provider’s implementation, when it comes to outsourcing, it will be difficult for the provider to create comprehensible bid invitations for subproviders.

Consequently, quality specification languages should support specification techniques which are independent from implementation details. As pointed out, this is not only required for customer orientation, but also aids providers in outsourcing processes. Additionally, in the LAL–brick, side independency is a first step towards the coverage of the user layer of abstraction.

Life cycle span Regarding the LAL–brick, the previous section has shown that not all quality specification techniques are qualified to cover a service’s full life cycle. As quality specification is already needed in the beginning (the *negotiation phase*), a quality description mechanism should try to cover the whole life cycle.

Especially the ability to reuse specification results should be addressed. This is desirable, as it would help providers in estimating feasibility of a customer’s demands during the negotiation phase. Second, it would aid providers in realizing services, as agreed quality could be more smoothly implemented during the provisioning phase. Third, for the usage phase, a life cycle spanning approach could help in measurement of quality characteristics. As a minimum requirement, specification techniques at the very least should point out which phase of the life cycle they were designed for.

Management functionality subject to quality As the MNM Service Model points out, *management functionality* is a vital part of any service, a point also reflected in the design of the LAL–brick. In fact, management functionality not only reports and manipulates quality aspects, but is also subject to quality considerations itself. This is even more important, as quality of management functionality can have influence on a service's usage quality.

For this, an example are situations when a main service is composed of subservices. Reporting of QoS from a subservice is part of its customer service management (CSM) functionality. When this reporting functionality has deficiencies, quality degradation from the subservice might not be determined by the main service. As a consequence its own usage functionality might be affected without being noticed, because quality degradations of the involved subservice are not noticed as the reporting functionality is degraded.

However, in current work the topic of applying quality to management functionality is not addressed. Although one can suppose that some tasks are similar to specifying quality of usage functionality, further research is needed. At least, specification techniques must be able to cope with the fact that in case of management functionality a different role (namely the *customer* instead of the *user*) is involved.

Awareness of Quality of Information (QoI) As the LAL–brick shows, a service's content may be a quality aspect, here referred to as the *information aspect*. Taking a web–based news service as an example, up–to–dateness of messages is, without a doubt, a quality criterion of the service. Dividing quality aspects of functionality (here: e.g. reachability of the service) from information quality (here: e.g. up–to–dateness and trustability of news) can aid over the whole service life cycle. During *negotiation* and *deployment*, *service agreements* with customers and subcontractors will be more accurate, outsourcing decisions gain a clearer basis. Naturally, technical infrastructure might influence the QoI. Regarding the news service, a certain up–to–dateness might require a different content management system or a faster communication infrastructure to subcontractors delivering news content. In the usage phase, e.g. in fault situations, root causes might be easier to find.

One might argue that this starts to involve high level semantics, a field which is hard to cope with. Nevertheless, separating quality aspects of a service's content from its functionality in fact already took place in some research areas. Context sensitive services are dealing with "Quality of Context" [HKLPR03,BKS03], for example the accuracy of location coordinates or temperature readings. Speaking in the terminology introduced by this paper, these are quality aspects of information. Future research in service management should be aware of this separation, should try to develop a generic approach and should try to invent techniques and mechanisms to incorporate and support Quality of Information (QoI).

It should be pointed out here that it would be unrealistic to demand or predict one single approach which is capable to span the whole LAL–brick and which can fulfill all of the requirements posed here. Instead, multiple approaches for different slices of the LAL–brick are more likely. But what should definitely be approached, is the interoperability between approaches and standards covering parts of the LAL–brick. These questions on interoperability are also subject for further research.

5 Conclusion and Outlook

In this paper a classification scheme for quality specification mechanisms and languages is presented. The classification emphasizes service orientation and consists of two parts. First a set of reference points is given, denoting the place in the MNM Service Model which is used to define quality properties of a service and on which later quality constraints are built upon. The second part of the classification scheme, called the LAL–brick, defines the dimensions along which quality description schemes can be classified. The classification scheme is applied to typical examples of current approaches in QoS specification. By this, the current state of the art in the field is outlined.

In the last part of the paper, observations of the classification scheme's application in conjunction with basic properties of the scheme itself are used to identify basic requirements and directions for future research in the field. Among others, one of the basic directions here is the awareness of the service life cycle. Additionally, a service content, or more abstract, the information it deals with, is also subject to quality (Quality of Information, QoI), which has to be separated from the quality of a service's usage and management functionality.

Further directions in our work include a specification scheme which focuses on the ability to reuse specification properties from preceding life cycle phases. Our second focus is targeted on the MNM Service Model. According to the results of this paper, it needs extensions regarding QoI, by that broadening its applicability to mobile and context aware service scenarios. Looking even further, approaches which claim the software development life cycle to be vital for quality specification (like [FK98,ZBS97]) must be investigated more precisely and eventually incorporated with the presented work.

Acknowledgment The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its web–server is located at http://wwwmnmteam.ifi.lmu.de/.

References

[Aag01] J. Ø Aagedal. *Quality of Service Support in Development of Distributed Systems.* Dr. scient. thesis, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, March 2001.

[BKS03] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context Information: What it is and why we need it. In *Proceedings of the 10th HP–OVUA Workshop*, volume 2003, Geneva, Switzerland, July 2003.

[COR04] Common object request broker architecture (corba/iiop). Specification version 3.0.2, OMG, March 2004.

[DR02a] G. Dreo Rodosek. *A Framework for IT Service Management.* Habilitation, Ludwig-Maximilians-Universität München, June 2002.

[DR02b] G. Dreo Rodosek. Quality Aspects in IT Service Management. In M. Feridun, P. Kropf, and G. Babin, editors, *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2002)*, Lecture Notes in Computer Science (LNCS) 2506, pages 82–93, Montreal, Canada, October 2002. IFIP/IEEE, Springer.

[FK98] Svend Frølund and Jari Koistinen. Qml: A language for quality of service specification. Report hpl-98-10, Software Technology Laboratory, Hewlett-Packard Company, September 1998.

[Flo96] Patrícia Gomes Soares Florissi. *QoSME: QoS Management Environment.* Phd thesis, Columbia University, 1996.

[GHH$^+$02] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. A Case–Driven Methodology for Applying the MNM Service Model. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th International IFIP/IEEE Network Operations and Management Symposium (NOMS 2002)*, pages 697–710, Florence, Italy, April 2002. IFIP/IEEE, IEEE Publishing.

[GHK$^+$01] M. Garschhammer, R. Hauck, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. The MNM Service Model — Refined Views on Generic Service Management. *Journal of Communications and Networks*, 3(4):297–306, December 2001.

[HAN99] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.

[HKLPR03] H.-G. Hegering, A. Küpper, C. Linnhoff-Popien, and H. Reiser. Management Challenges of Context–Aware Services in Ubiquitous Environments. In *Self–Managing Distributed Systems; 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003, Heidelberg, Germany, October 2003, Proceedings*, number LNCS 2867, pages 246–259, Heidelberg, Germany, October 2003. Springer.

[ITU97] Open Distributed Processing – Interface Definition Language. Draft Recommendation X.920, ITU, November 1997.

[JN04] Jingwen Jin and Klara Nahrstedt. QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy. In *IEEE Multimedia Magazine*, to apear 2004.

[PJS$^+$00] P Pal, Loyall J., R. Schantz, J. Zinky, R. Shapiro, and J. Megquier. Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration. In *Proceedings of ISORC 2000, The Third IEEE International Symposium on Object-Oriented Real-time Distributed Computing*, Newport Beach, CA., March 2000.

[ZBS97] J. Zinky, D. Bakken, and R. Schantz. Architectural Support for Quality of Service for CORBA Objects. In *Theory and Practice of Object Systems*, January 1997.