Formalization of Network Quality-of-Service Requirements

Christian Webel and Reinhard Gotzhein

Computer Science Department, University of Kaiserslautern, Kaiserslautern, Germany
{webel,gotzhein}@informatik.uni-kl.de

Abstract. The provision of *network Quality-of-Service* (network QoS) is a major challenge in the development of future communication systems. Before designing and implementing these systems, the network QoS requirements are to be specified. Existing approaches to the specification of network QoS requirements are mainly focused on specific domains or individual system layers. In this paper, we present a holistic, comprehensive formalization of network QoS requirements, across layers. QoS requirements are specified on each layer by defining QoS domain, consisting of QoS performance, reliability, and guarantee, and QoS scalability, with utility and cost functions. Furthermore, we derive preorders on multi-dimensional QoS domains, and present criteria to reduce these domains, leading to a manageable subset of QoS values that is sufficient for system design and implementation. The relationship between layers is formalized by two kinds of QoS mappings. QoS domain mappings associate QoS domains of two abstraction levels. QoS scalability mappings associate utility and cost functions of two abstraction levels. We illustrate our approach by examples from the case study *Wireless Video Transmission*.

1 Introduction

One of the major challenges in communication networks is the provision of *network quality of service* (network QoS). By network QoS, we refer to the degree of well-definedness and controllability of the behaviour of a communication system with respect to quantitative parameters [1]. The need for network QoS arises from the fact that, for state-of-the-art distributed applications, it is essential that they offer their functionality with specified performance, reliability, and guarantee. In addition, communication systems and applications have to adapt to varying traffic and channel quality. The realization of such adaptive behaviour can in fact be seen as one of the key challenges in the development of communication systems supporting *network quality of service*. It requires a cross-layer approach with suitable abstractions and mappings between resource views of different layers.

Our current work aims at establishing a holistic engineering approach for communication systems, including network QoS provision. In this context, we are

J. Derrick and J. Vain (Eds.): FORTE 2007, LNCS 4574, pp. 309–324, 2007.

investigating techniques for the formal specification of QoS requirements on different system levels. Existing techniques are mainly focused on specific domains or system layers. In this paper, we present a formalization of network QoS requirements across layers. The formal relationship between layers is established by QoS domain mappings. To formalize scalability, we define utility and cost functions on each layer, which are used to derive preorders on QoS domains. Utility and cost functions of different layers are related by QoS scalability mappings. To achieve consistency between these functions of different layers, QoS scalability mappings are derived from QoS domain mappings.

The remaining part of the paper is organized as follows: In Section 2, we survey related work. In Section 3, we present our formalization and specification of network QoS requirements. Section 4 describes the different abstraction levels in communication systems supporting QoS and the mappings between these levels. Section 5 illustrates the approach by a case study. Last, Section 6 presents conclusions and future work.

2 Related Work

To cope with various requirements of system designs, user preferences, middleware, hardware, networks, operating systems, and applications, several QoS specification techniques have been proposed (see [2] for a classification):

- QML (Quality Modelling Language) [3] is focused on the specification of application layer QoS requirements. QoS requirements of lower layers, QoS scaling, and QoS mappings are not addressed.
- CQML [4] adopts some of the fundamental concepts of QML, and also addresses dynamic QoS scaling. As QML, it is focused on the application layer.
- QDL (Quality Description Language) has been proposed as a part of the QuO (Quality Objects) framework [5] that supports QoS on the CORBA object layer. With QDL, it is possible to specify QoS requirements on application layer and on resource layer, and to define QoS scaling.
- The Quality Assurance Language (QuAL) is part of QoSME [6]. With QuAL, QoS requirements are specified in a process-oriented way. The Quality-of-Service Architecture (QoS-A) [7] uses a parameter-based specification approach.
- In [8], an approach for specifying and mapping QoS in distributed multimedia systems is presented. Based on the specification, fuzzy-control is used for QoS scaling.
- Formal QoS mappings have not been studied thoroughly so far. Some partial results can be found in [9] and [10].

In summary, it can be stated that previous formal treatments of QoS address only some aspects of QoS requirement specification, focusing, for instance, on a subset of abstraction layers, or leaving out QoS mappings. Our work comprises the aforementioned issues and therefore provides a holistic, comprehensive formalization of network QoS requirements, across layers.

3 Formalization of Network Quality of Service

The need for *formalization* of network quality of service arises from the fact that a *precise* description of network QoS between service user and service provider is needed to police, control, and maintain the data flow a user emits to the communication system. Further on, the mechanisms realizing these functionalities require a *precise* QoS description. These mechanisms are typically integrated across layers; therefore, more than one viewpoint on the network QoS requirements is needed. To rigorously relate these viewpoints, formal QoS mappings are to be defined. In this section, we start to formalize network QoS by defining *QoS domain*, *QoS scalability* and *QoS specification*. The formal definition of QoS mappings will be addressed in Sect. 4.

3.1 QoS Domain

The *QoS domain* captures the QoS characteristics of a class of data flows, i.e. performance, reliability, and guarantee:

Definition 1 (QoS Domain). *The QoS domain Q is defined as $Q = P \times R \times G$, where P is the performance domain, R is the reliability domain, and G is the guarantee domain. $q = (p, r, g)$ denotes an element of Q, called QoS value.*

QoS performance describes efficiency aspects characterizing the required amount of resources and the timeliness of the service (e.g., peak and average throughput, delay, jitter, burst characteristics). The relevant aspects are included in the *QoS performance domain* P, which we formalize as follows:

Definition 2 (QoS Performance). *A QoS performance domain P is defined as $P = P_1 \times \ldots \times P_n = \prod_{i=1}^{n} P_i$, where $P_1, \ldots, P_n$ are performance subdomains.*

QoS reliability describes the *safety-of-operation* aspects characterizing the fault behaviour (e.g., loss rate and distribution, corruption rate and distribution, error burstiness). It can significantly impact the overall throughput and functionality on lower system layers, since it requires redundancy (e.g., retransmission, forward error control). The relevant aspects are included in the *QoS reliability domain* R:

Definition 3 (QoS Reliability). *The QoS reliability domain R is defined as $R = Loss \times Period \times Burstiness \times Corruption$, with $Loss = \mathbb{N}_0$, $Period = \mathbb{R}_+$, $Burstiness = \mathbb{R}_+$, and $Corruption = \{r \in \mathbb{R} \mid 0 \leq r < 100\}$.*

Reliability addresses *loss* corresponding to a layer-specific data unit (e.g. picture frames or lower system layer PDUs), the *period* in which data loss occurs, and the *burstiness*, i.e. the duration of a successional appearance of data loss. As a fourth parameter, the permitted *corruption rate* for a layer specific data unit in percent is given.

QoS guarantee describes the degree of commitment characterizing the binding character of the service. Four degrees of commitment are distinguished. *Best-effort* denotes the minimal degree, meaning that no guarantees are given. *Deterministic* refers to the highest degree, meaning that hard guarantees are provided.

Statistical expresses that guarantees are given with a specified probability. Finally, *enhanced best-effort* denotes better-than-best-effort guarantees: in periods of sufficient resources, statistical or deterministic guarantees are provided; otherwise, a priority-based best-effort scheme is used. QoS guarantee is formalized by the *QoS guarantee* domain:

Definition 4 (QoS Guarantee). *The domain of QoS guarantee G is defined as $G = DoC \times Stat \times Prio$, where $Stat = \{p \in \mathbb{R} \mid 0 < p \leq 1\}$, $Prio = \mathbb{N}$, and $DoC = \{bestEffort, enhancedBestEffort, statistical, deterministic\}$.*

The guarantee consists of a degree of commitment *DoC*, a corresponding value *Stat* in case of statistical guarantees, and a *priority*. The priority determines the relative importance between two or more QoS requirements (traffic contracts).

3.2 QoS Scalability

Varying communication resources require adaptive mechanisms to avoid network overload, and to scale the application service. The *QoS scalability* S describes the control aspects characterizing the scope for a dynamic adaptation of the QoS aspects of a data flow (described by a QoS domain) to a certain granted network quality of service:

Definition 5 (QoS Scalability). *Let Q be a QoS domain. The domain of QoS scalability S is defined as $S = Util \times Cost \times Up \times Down$, where $Util = \{u \mid u : Q \to [0,1]\}$, $Cost = \{c \mid c : Q \to \mathbb{R}_+\}$, and $Up, Down \in \{x \in \mathbb{R}_+ \mid 0 \leq x \leq 1\}$.*

The elements of *Util* and *Cost* are called *utility functions* and *cost functions*, respectively. A utility function determines the usefulness of QoS values $q \in Q$. This information is crucial for upscaling and downscaling, and has to be provided on all system layers. The utility of QoS values depends on the application scenario, but not necessarily on the amount of needed resources. The latter is expressed by the cost function, which can be tailored to the actual resource situation, associating higher costs with scarcer resources. In other words, given two QoS values q and q' with $u(q) > u(q')$, it is possible that $c(q) < c(q')$, i.e., q consumes less resources than q'. We will have to take this into account when defining QoS scalability mappings (see Sect. 4.3). Related to the utility function, two values $up \in Up$ and $down \in Down$ are used to define thresholds for up- and downscaling, i.e. a scaling is only performed, if the benefit for the user increases/decreases more than $up/down$ percent.

According to [8], the utility function u can be defined using functions on P, R and G:

$$u_P : P \to [0,1], \quad u_R : R \to [0,1], \quad u_G : G \to [0,1] \tag{1}$$

A possible definition u for a QoS value $q = (p, r, g)$ is:

$$u(q) = \min\{u_P(p) \cdot w_P, u_R(r) \cdot w_R, u_G(g) \cdot w_G\} \tag{2}$$

This definition emphasizes that usually, a minimum benefit of each of the QoS value constituents is required. Other definitions can be given by introducing

weights w_P, w_R, and w_G, reflecting the relative importance of performance, reliability, and guarantee, respectively, in the current application scenario, with $u(q)$ being the sum of the weighted constituents of q. In both cases, the result of (2) has to be normalized into the interval $[0, 1]$ (see Definition 5).

The utility function u (the cost function c) induces an *equivalence relation* $\sim_u$ ($\sim_c$) and a preorder $\lesssim_u$ ($\lesssim_c$) on the QoS domain Q:

$$\sim_u =_{\text{DF}} \{(q_1, q_2) \in Q \times Q \mid u(q_1) = u(q_2)\} \tag{3}$$

$$\lesssim_u =_{\text{DF}} \{(q_1, q_2) \in Q \times Q \mid u(q_1) \leq u(q_2)\} \tag{4}$$

In certain scenarios, several QoS values may have the same usefulness according to the utility function u. For instance, a user may not be able to distinguish between 25 and 26 picture frames per second, and therefore assigns the same utility value to both QoS values. For this reason, $\lesssim_u$ is a preorder on Q in general. Based on $\sim_u$ ($\sim_c$), we define u-equivalence (c-equivalence) classes of Q:

$$[x]_u = \{q \in Q \mid q \sim_u x\} \tag{5}$$

These definitions form the basis for consistency criteria of QoS mappings introduced in Sect. 4.

Apart from defining the utility of QoS values, the actual costs are required in order to provide the scope for dynamic adaptation. For instance, it is possible that for QoS values q, q', and q'', $u(q) > u(q') > u(q'')$, while the costs in terms of resources are $c(q') > c(q) > c(q'')$. Assume that q'' is currently provided, and the resource situation improves. In this case, it is certainly better to directly scale to q, omitting q'. This means that although q' has a utility in-between q and q'', it should not be used. This observation can be exploited such that for a given utility, the QoS value with minimum cost is selected. For each u-equivalence class, we keep one representative value with minimum cost (Step 1). Next, we observe that in general, while the utility increases, the cost may decrease. Therefore, some u-equivalence classes become obsolete, as it would be better to skip some QoS values to get even better utility for less cost (Step 2). These ideas are formalized in the following definitions.

To formalize Step 1 (keeping one representative per u-equivalence class with minimum cost), we define the *reduced* QoS domain Q^u by selecting the best element of each u-equivalence class of Q regarding c. Let m be the cardinality of $Q/\!\sim_u$, the quotient set of Q w.r.t. $\sim_u$, and let $[x]^i_u$ denote the ith element of $Q/\!\sim_u$ regarding $\lesssim_u$ (ith u-equivalence class). Then,

$$Q^u = \{q_1, \ldots, q_m\} \cap Q', \quad q_i = q \in [x]^i_u \mid \forall y \in [x]^i_u \,.\, q \lesssim_c y, \quad 1 \leq i \leq m \tag{6}$$

Q^u contains elements in the specified subset Q' of a QoS domain Q (see Sect. 3.3, (8)) and is totally ordered by $\lesssim_u$.

To formalize Step 2 (discarding of QoS values with higher cost, but less utility), we define the derived QoS domain $Q^{u,c}$ as follows:

$$Q^{u,c} = \{q \in Q^u \mid \forall y \in Q^u \,.\, c(q) > c(y) \Rightarrow u(q) > u(y)\} \tag{7}$$

3.3 Specification of Network QoS Requirements

A *QoS requirements specification* captures the concrete QoS requirements on one system layer by defining the set of valid QoS domain values and a QoS scalability value. The specification is used to configure, manage and maintain QoS mechanisms located on each system layer.

Definition 6 (QoS Requirements Specification). *Let Q be a QoS domain and S be a QoS scalability domain. A QoS requirements specification qosReq is defined as a triple (q_{min}, q_{opt}, s), where $q_{min},\ q_{opt} \in Q$ and $s \in S$.*

The QoS values q_{min} and q_{opt} specify the set $Q' \subseteq Q$ of valid QoS domain values. To obtain Q' from q_{min} and q_{opt}, the preorder $\lesssim_u$ induced by the utility function (see (4)) is applied:

$$Q' = \{q \in Q \mid q_{min} \lesssim_u q \lesssim_u q_{opt}\} \tag{8}$$

4 QoS Mappings

So far, we have introduced the formalization and specification of network QoS requirements. Such requirements can be specified from different viewpoints, on different levels of abstraction. To relate QoS requirements of different levels, QoS mappings are needed. The reason for that is that QoS management tasks are typically embedded in the communications system, prevalent across layers, hiding complex tasks from the application. This leads to simple QoS specifications on higher system layers, whereas on lower system layers, the complexity increases.

In this section, we will start by identifying several levels of abstraction, and will then formally define mappings between QoS domains and QoS scalability.

4.1 QoS Abstraction Levels

To implement network QoS requirements, QoS mechanisms on several system layers are needed, each with its own viewpoint describing the data flow traversing the (communication) system. We call these different viewpoints *QoS Abstraction Levels*:

1. *User Level:* From the user's point of view, the application is characterized in terms of scenarios, e.g., *surveillance* or *panorama*.
2. *Application Level:* From the user scenarios, the system developer derives the network QoS requirements of the application level. QoS parameters are application-specific, e.g., picture frame rate, or JPEG quality.
3. *Communication Level:* QoS parameters of the communication level characterize the data flow in terms of, e.g., transmission units, transmission periods, transmission delay, delay jitter. On this level, the QoS requirements are still platform-independent.

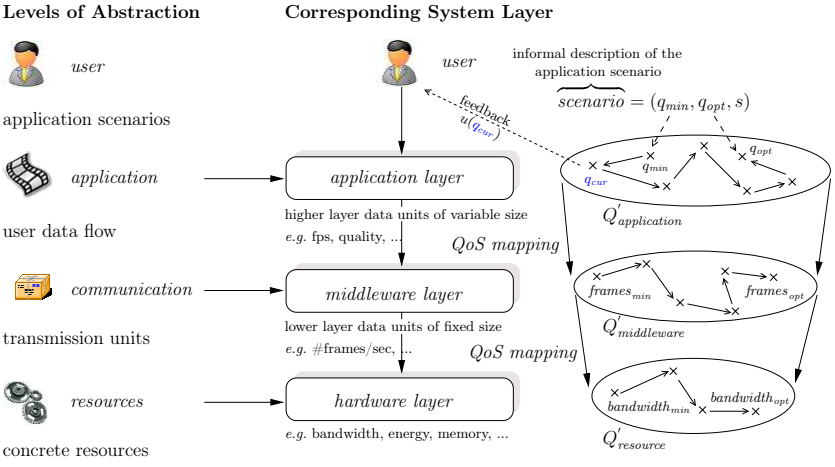

Fig. 1. Abstraction Levels and QoS Mappings

4. *Resource Level:* On resource level, the QoS requirements are specified in terms of concrete hardware parameters, e.g., bandwidth, energy, cpu cycles, memory. This specification is platform-specific.

The abstraction levels are shown in Fig. 1 (left), and associated with a specific system layer (middle). Network QoS requirements are specified on all abstraction layers, expressing the particular viewpoint. To implement the requirements, these viewpoints have to be related. For instance, QoS performance of the video application – stated in terms of resolution, JPEG quality, and picture frames per second – must be related to QoS performance of the communication level – stated in terms of number of data frames per picture frame and period. We introduce two kinds of QoS mappings to formalize this relationship, the QoS domain mapping, and the QoS scalability mapping.

4.2 QoS Domain Mapping

To relate QoS requirements of different abstraction levels, QoS mappings are needed. In this section, we introduce *QoS domain mappings* between a higher layer QoS domain Q_h and a lower layer domain Q_l. This is illustrated in Fig. 1 (right).

Definition 7 (QoS Domain Mapping). *Let Q_h, Q_l be QoS domains on different system layers. A QoS domain mapping $dm : Q_h \rightarrow Q_l$ is a function from a (higher layer) QoS domain Q_h to a (lower layer) QoS domain Q_l. The domain mapping dm may be defined using auxiliary functions as follows:*

$$dm_P : Q_h \rightarrow P_l \text{ (performance mapping)}$$
$$dm_R : Q_h \rightarrow R_l \text{ (reliability mapping)}$$
$$dm_G : Q_h \rightarrow G_l \text{ (guarantee mapping)}$$

In general, QoS mappings are neither injective nor surjective. This means that two different QoS values $q_1, q_2 \in Q_h$ could be mapped to the same $q_l \in Q_l$, and that the values of dm do not span the whole codomain Q_l. For these reasons, the mapping of the scalability requirements specification, especially the utility function, is nontrivial. In the following, we elaborate on the three auxiliary functions.

QoS Performance Mapping. The QoS performance mapping dm_P translates the performance parameters into each other. The performance parameters are system layer and hardware dependent, i.e. parameters like the maximum transfer unit (MTU), the path MTU, or the frame format have to be considered.

Definition 8 (QoS Performance Mapping). *Let P_h, P_l be performance domains on different system layers. A QoS performance mapping $dm_P : P_h \rightarrow P_l$ is a function translating performance values $p_h \in P_h$ into new values $p_l \in P_l = P_{l_1} \times \cdots \times P_{l_n}$. To define dm_P, auxiliary functions $dm_{P_i}(p_h) = p_{l_i},\ \forall\, i \leq l_n$ can be used.*

QoS Reliabiliy Mapping. Higher layer transmission units (e.g., picture frames) can be larger than lower layer units and therefore have to be fragmented and reassembled. This, however, complicates the definition of the QoS reliability mapping (see [11]). To illustrate this, consider the example in Fig. 2. On application layer, the variable-size picture frames are fragmented into maximum-size middleware packets. On middleware layer, we assume a loss ratio of 30%. The loss can be caused by packet loss, corrupted, dropped, or late-delivered PDUs. Further, we assume that a loss of even one lower layer packet results in the loss of the entire picture frame. In Figure 2.a, the loss ratio results in a picture frame loss of 33%. If the loss is uniformly distributed (as shown in Fig. 2.b), the same ratio leads to a loss on application layer of 100%.

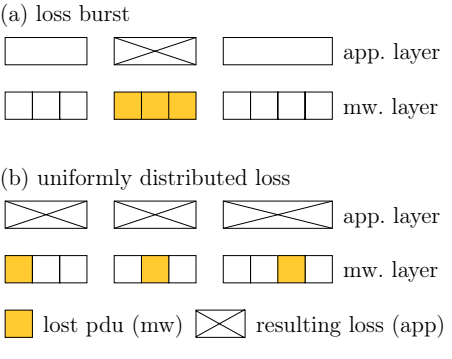

Fig. 2. Upper layer PDUs vs. lower layer PDUs

Notice that a simple description of the lower layer loss or corruption probability is not sufficient for deriving the expected upper layer reliability parameters.

Moreover, uniformly distributed losses may be more adverse than bursty losses. To define the QoS reliability mapping, a segmentation model of the user data is needed. In our case study, this model would introduce probability distributions of picture frame sizes and resulting probability mass functions of the number of needed middleware packets. Further, an error models characterizing the loss and/or corruption process is needed. This error model strongly depends on the chosen base technology. The definition of segmentation and error model are out of the scope of this paper. A treatment of these aspects can be found in [11].

QoS Guarantee Mapping. The function dm_G maps the guarantees specified on one system layer to corresponding guarantees on another. Ideally the guarantees should stay the same during a mapping process. But in exceptional cases, e.g., if the underlying base technology does not support required degree of commitment, an upgrade is permitted. For example, a mapping from *statistical* to *deterministic* guarantees is always feasible, whereas a mapping vice versa could result in a violation of the traffic contract.

4.3 QoS Scalability Mapping

QoS scalability describes the control aspects characterizing the scope for dynamic adaptation of QoS parameters. To apply scaling on different levels of abstraction, a QoS scalability mapping is needed. For consistency, this mapping has to ensure that the utility of QoS values of different abstraction levels that are related by the QoS domain mapping dm is the same. To enforce this consistency condition, we will now define a scalability mapping such that, given a utility function u_h, yields the corresponding utility function u_l. Next, we will introduce a *cost function* that associates costs with QoS values. Based on this cost function, we will finally arrive at a reduced set of QoS values characterizing the actual scope for dynamic adaptation.

In the following definition, let $Q_l^* = \{q_l \in Q_l \mid \exists\, q_h \in Q_h \,.\, dm(q_h) = q_l\}$ denote the set of mapped QoS values, and let $[q_h]_{dm} = \{x \in Q_h \mid x \sim_{dm} q_h\}$, $\sim_{dm} = \{(q_h, q_h^{'}) \in Q_h \times Q_h \mid dm(q_h) = dm(q_h^{'})\}$, denote the equivalence classes containing those QoS values q_h that are mapped to the same q_l.

Definition 9 (QoS Scalability Mapping). *Let S_h, S_l be scalability domains on different system layers. A QoS scalability mapping is a set of four mapping functions sm_{Util}, sm_{Cost}, sm_{Up} and sm_{Down}, translating the different scalability domains into each other (see Fig. 3):*

$$
\begin{aligned}
sm_{Util} : Util_h \rightarrow Util_l;\ \forall u_h \in Util_h \,.\, \forall q_l \in Q_l^* \,.\, u_l(q_l) =_{DF} u_h(q_h) \mid \\
q_h \in Q_h \wedge dm(q_h) = q_l \wedge \forall x \in [q_h]_{dm} \,.\, u_h(q_h) \geq u_h(x) \\
sm_{Cost} : Cost_l \rightarrow Cost_h;\ \forall c_l \in Cost_l \,.\, \forall q_h \in Q_h \,.\, c_h(q_h) =_{DF} c_l(dm(q_h)) \\
sm_{Up} : Down_h \rightarrow Up_l; \forall x \in Up_h \,.\, sm_{Up}(x) =_{DF} x \\
sm_{Down} : Down_h \rightarrow Down_l; \forall x \in Down_h \,.\, sm_{Down}(x) =_{DF} x
\end{aligned}
$$

Some explanations are in order. Let $q_h, q_h^{'} \in Q_h,\ u_h(q_h) > u_h(q_h^{'}),\ dm(q_h) = dm(q_h^{'})$. In other words, although the utility of q_h is higher than that of $q_h^{'}$, they

$$\begin{array}{lrcl} \textit{higher layer:} & S_h & = & \textit{Util}_h \times \textit{Cost}_h \times \textit{Up}_h \times \textit{Down}_h \\ & & & \downarrow sm_{Util} \quad \uparrow sm_{Cost} \quad \downarrow sm_{Up} \quad \downarrow sm_{Down} \\ \textit{lower layer:} & S_l & = & \textit{Util}_l \times \textit{Cost}_l \times \textit{Up}_l \times \textit{Down}_l \end{array}$$

Fig. 3. Scalability mapping

consume the same amount of resources $q_l = dm(q_h)$. In this case, the utility of q_l is chosen as $u_l(q_l) =_{\text{DF}} u_h(q_h)$, i.e. the better value. This means, that when the resources q_l are available, they are exploited as best as possible. This idea is generalized in the definition of the mapping function sm_{Util}, where to each value of $q_l \in Q_l^*$, the maximum utility of all corresponding values $q_h \in Q_h$ is assigned. Note that costs are mapped from lower to higher system layer and that the thresholds for upscaling and downscaling remain unmodified by the QoS scalability mapping.

With the QoS mappings *dm*, *sm* and the reduced QoS domain (see (7)), it is possible to define a *scaling function* to be used in system design and implementation. A scaling function $scal_{u,c_l} : Q_l \to Q^{u,c_h} \cup \{0\}$ maps a lower layer QoS values describing the currently granted network QoS to a higher layer *cost-optimal* QoS value. The function selects the best possible, i.e. the *optimum* QoS value $q \in Q^{u,c_h}$ regarding the utility function u in compliance with the currently granted QoS resources $q_{granted} \in Q_l$, if such an element exists, otherwise 0. For this reason, the cost function c_l has to be mapped to a corresponding higher layer cost function c_h in order to properly reduce Q. The scaling function is defined as follows:

$$scal_{u,c_l}(q_{granted}) = \max_u\{q \in Q^{u,sm_{Cost}(c_l)} \mid c_l(dm(q)) \leq c_l(q_{granted})\} \quad (9)$$

whereas the maximum operator max_f for a given set X defines x as an f-maximal element of X iff $x \in X$ and $\forall y \in X : (f(x) \leq f(y) \Rightarrow x = y)$, short $\max_f\{X\}$. The maximum of an empty set is defined as zero, i.e. $\max_f \varnothing = 0$.

5 Case Study *Wireless Video Transmission*

We illustrate the formalization of network QoS by the application *Wireless Video Transmission*, which is used in our remotely controlled airship [12]. The quality of video transmission as perceived by the user depends on picture frame resolution, JPEG compression rate, and picture frame rate. On communication layer, this translates to the number of messages per picture frame, message rate and delay, and finally to channel bandwidth and delay.

In this application (see Fig. 4), we distinguish two usage scenarios, *surveillance* for movement detection and *panorama* for landscape recording. Given the QoS domain Q_{video}, the network QoS requirements $qosReq_{sur}$ and $qosReq_{pan}$ are defined by triples, consisting of optimal and minimal QoS values, and a QoS scalability value. From these triples, the subsets *surveillance* and *panorama* of Q_{video} are determined, applying the preorder $\lesssim_u$ induced by the utility function.

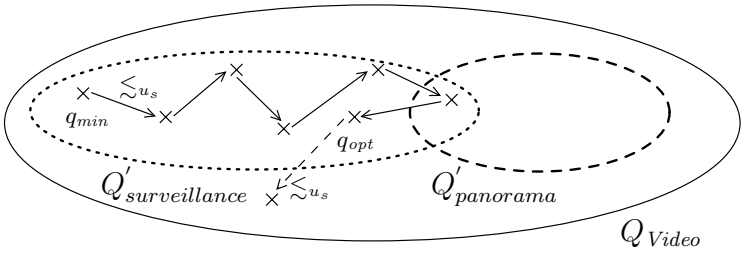

Fig. 4. QoS Requirements Specification

5.1 QoS Domain

A QoS domain Q is specified by defining concrete subdomains performance P, reliability R, and guarantee G. As an example, we define subdomains for the QoS domain Q_{video}, and concrete QoS requirement specifications $qosReq_{sur}$ and $qosReq_{pan}$.

The quality of video transmission depends on picture frame resolution, JPEG compression rate, and picture frame rate. Further QoS parameters are transmission delay and delay jitter, which we omit in the following. For our case study, the concrete domains on application layer are $P_1 = Resolution$, $P_2 = Quality$, $P_3 = FrameRate$, yielding P_{video}.

$$\begin{aligned} P_{video} &= Resolution \times Quality \times FrameRate \\ Resolution &= \{(320, 240), (480, 360), (640, 480)\} \\ Quality &= \{25, 50, 75\} \\ FrameRate &= \{f \in \mathbb{N} \mid 1 \leq f \leq 25\} \end{aligned} \tag{10}$$

Typical element of P_{video} is $p = (\,(res_x, res_y), qual, fps\,)$. An appropriate specification of the required performance for surveillance purposes is given by

$$p_{minSur} = ((320, 240), 25, 10), \quad p_{optSur} = ((640, 480), 75, 20)\,. \tag{11}$$

The reliability specification identifies concrete values for loss, period, burstiness, and corruption (see Definition 3). For the video transmission, we define $r_{minSur} = r_{optSur} = (3, 1, 2, 0)$, specifying a permitted data loss of *three* picture frames per *one* second, loss bursts of up to *two* picture frames, and a corruption rate of *zero* percent.

A guarantee specification is given by $g_{minSur} = g_{optSur} = (enhancedBestEffort, 0.8, 8)$. If due to the current resource situation only priority best-effort guarantees can be provided, the priority of 8 enables the wireless video transmission to gain privilege over other applications with lower priorities (< 8). If adequate statistical guarantees are offered, a minimum of 80 percent is required.

5.2 QoS Scalability

To specify QoS scalability, concrete utility functions u_P, u_R, u_G, cost function c, and two thresholds *up* and *down* are to be defined. Due to limitations of

space, we omit the specification of cost functions, which in principle are similar in style to the utility functions. For the video transmission, we start by defining auxiliary functions for each performance subdomain, normalizing the utility of each parameter to a value in $[0, 1]$:

$$\begin{aligned} u_{res} &: \mathit{Resolution} \rightarrow [0,1], \;\; u_{res}(res) &&= \tfrac{res_x - 160}{480} \\ u_{qual} &: \mathit{Quality} \quad\;\; \rightarrow [0,1], \;\; u_{qual}(qual) &&= \tfrac{qual}{75} \\ u_{fps} &: \mathit{FrameRate} \rightarrow [0,1], \;\; u_{fps}(fps) &&= \tfrac{fps}{25} \end{aligned} \tag{12}$$

Next, we define weights reflecting the relative importance of each subdomain corresponding to the current application scenario. For instance, picture frame rate is the decisive video parameter in case of surveillance, while resolution and quality are of particular importance in the panorama scenario. With the weights $\omega_{res} = 0.1$, $\omega_{qual} = 0.1$, $\omega_{fps} = 0.8$ for surveillance and $\upsilon_{res} = 0.4$, $\upsilon_{qual} = 0.4$, $\upsilon_{fps} = 0.2$ for panorama, we obtain the following performance utility functions:

$$\begin{aligned} u_{P_{sur}} &: P_{video} \rightarrow [0,1], \quad u_{P_{sur}} = 0.1 \cdot u_{res} + 0.1 \cdot u_{qual} + 0.8 \cdot u_{fps} \\ u_{P_{pan}} &: P_{video} \rightarrow [0,1], \quad u_{P_{pan}} = 0.4 \cdot u_{res} + 0.4 \cdot u_{qual} + 0.2 \cdot u_{fps} \end{aligned} \tag{13}$$

Since $r_{min} = r_{opt}$ and $g_{min} = g_{opt}$ in both QoS requirement specifications, the utility subfunctions operating on R and G can be defined as follows:

$$u_G(x) = \begin{cases} 0 & \text{if } x < g_{min} \\ 1 & \text{otherwise} \end{cases}, \quad u_R(x) = \begin{cases} 0 & \text{if } \frac{l}{p} > \frac{l_{min}}{p_{min}} \vee b > b_{min} \vee c > c_{min} \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

u_G implies an order on the guarantee domain that can be intuitively given by arranging the values (1) according to their degree of commitment (*bestEffort* to *deterministic*), then (2) according to the statistical component and last (3) according to their priority. The parameters l, p, b, and c in the definition of u_R refer to loss, period, burstiness, and corruption, respectively.

Inserting into (2) yields the following utility functions on Q_{video}:

$$u_{sur}(q) = \min\{u_{P_{sur}}(p), u_R(r), u_G(g)\}, \quad u_{pan}(q) = \min\{u_{P_{pan}}(p), u_R(r), u_G(g)\} \tag{15}$$

In both cases, downscaling should be performed if the benefit decreases by 10 percent and upscaling should only be done if the benefit increases by 20 resp. 10 percent, leading to the following complete specification of the scalability requirements:

$$s_{sur} = (u_{sur}, c_{sur}, 0.2, 0.1), \qquad s_{pan} = (u_{pan}, c_{pan}, 0.1, 0.1) \tag{16}$$

Based on the utility functions, the QoS values are divided into equivalence classes. Table 1 lists some $u_{P_{sur}}$-equivalence classes of P_{video}. In order to minimize the overall number of classes, the utility has been rounded to two decimal places, resulting in a reduction from 125 to 44 classes.

In Figure 5, the QoS domain is reduced, applying Steps 1 and 2 as defined in Sect. 3.2. The utility function u_{sur} partitions Q_{video} into 45 u_{sur}-equivalence

Table 1. $u_{P_{sur}}$-equivalence classes of P_{Video}

utility	$u_{P_{sur}}$**-equivalence class**
0.1	((320,240),25,1)
0.13	((480,360),25,1) ((320,240),25,2) ((320,240),50,1)
...	...
0.39	((320,240),25,10) ((640,480),75,6) ... ((480,360),25,9) ((480,360),50,8)
0.42	((640,480),25,9) ((480,360),25,10) ... ((320,240),25,11) ((480,360),75,8)
...	...
0.84	((640,480),25,22) ((640,480),75,20) ... ((480,360),25,23) ((320,240),75,22)
...	...
1.0	((640,480),75,25)

classes (rounded to two decimal places). Since the result of u_R resp. u_G could be 0, the overall number of equivalence classes increases by one (cf. $u_{P_{sur}}$-equivalence classes). If all values of the QoS domain Q are arranged along the x-axis, respecting the preorder $\lesssim_u$, then the resulting graph is a monotonically increasing step function. In addition, a cost function c is depicted in Fig. 5, describing the needed resources on lower system layer. Note that the costs basically increase with the utility, however, within a given u_{sur}-equivalence class, different costs may be associated with QoS values having the same utility. The reduced domain Q^u is formed by selecting the cost-optimal QoS values out of each u_{sur}-equivalence class (see Table 2) and intersecting this selection with $Q^{'}$, leading to $Q^u = \{q^u_{16}, \ldots, q^u_{39}\}$. Step 2 (cf. (7)) induces a further reduction of the overall number of QoS values, since for example q^u_{37} can be omitted due to the higher cost but less utility compared to q^u_{38}. This leads to $Q^{u,c}$ with a total number of 16 QoS values.

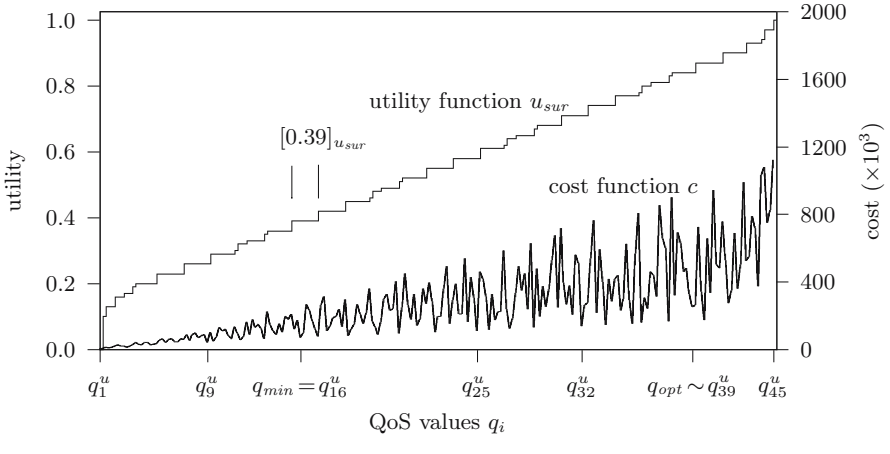

Fig. 5. Reduction of Q

Table 2. Cost-optimal QoS values

utility	QoS value		cost
0.0	q_1^u	$(((320,240),25,1),\ r_{minSur},\ <g_{minSur})$	7000
0.1	q_2^u	$(((320,240),25,1),\ r_{optSur},\ g_{optSur})$	7000
0.13	q_3^u	$(((320,240),50,1),\ r_{optSur},\ g_{optSur})$	11000
...	...	...	...
0.39	$q_{min} = q_{16}^u$	$(((320,240),25,10),\ r_{optSur},\ g_{optSur})$	70000
0.42	q_{17}^u	$(((320,240),25,11),\ r_{optSur},\ g_{optSur})$	77000
...	...	...	...
0.81	q_{37}^u	$(((320,240),75,21),\ r_{optSur},\ g_{optSur})$	315000
0.83	q_{38}^u	$(((320,240),25,24),\ r_{optSur},\ g_{optSur})$	168000
0.84	$q_{opt} \sim q_{39}^u$	$(((320,240),50,23),\ r_{optSur},\ g_{optSur})$	253000
...	...	...	...
1.0	q_{45}^u	$(((640,480),75,25),\ r_{optSur},\ g_{optSur})$	1125000

5.3 QoS Mapping

The QoS performance mapping from the application layer performance domain P_{video} to an underlying middleware layer with $P_{mw} = \#Frames \times Period$ is formally defined as follows:

$$dm_P : P_{video} \rightarrow P_{mw}$$
$$dm_P((res_x, res_y), fps, quality) = (\#frames, period), \text{ with}$$
$$dm_{P_1}((res_x, res_y), fps, quality) = \#frames = \left\lceil \frac{(160 \cdot quality + 3000) \cdot (res_x - 160)/160}{payload\ \ bytes\ \ per\ \ frame} \right\rceil$$
$$dm_{P_2}((res_x, res_y), fps, quality) = period = \frac{1}{fps}$$

On application layer, QoS performance is defined by resolution, picture frames per second, and quality. On middleware layer, we have the number of data frames required for the transmission of one picture frame, and the period between two picture frames, i.e. a burst of data frames.

The QoS reliability mapping is to be based on segmentation and error models, which are outside the scope of this paper, and therefore omitted. For the QoS guarantee mapping, we assume that the guarantees specified on higher levels are supported by the base technology, so that the guarantees can be maintained across layers.

The QoS scalability mapping is universally defined in Definition 9, independent from application, system, and hardware. Therefore, no specific mapping is needed.

6 Conclusion and Future Work

In this paper, we have presented a holistic, comprehensive formalization of network QoS requirements, across layers. QoS requirements are specified on each layer by defining a multi-dimensional QoS domain and QoS scalability. Based on

these definitions, we have derived preorders on multi-dimensional QoS domains, and have presented criteria to reduce these domains to manageable subsets, sufficient as a starting point for system design and implementation. To formally relate layers, we have introduced two kinds of QoS mappings, called QoS domain mappings and QoS scalability mappings.

All formalizations so far are based on mathematics. For better usability, we intend to define a formal QoS requirement specification language, with intuitive keywords and structuring capabilities. This language should be powerful enough to host the concepts and criteria we have introduced in this paper. Also, the language should be supported by tools that can, for instance, construct QoS mappings as far as they have been defined in this work.

Another step is to specify designs that satisfy given QoS requirement specifications. In particular, there is need for defining a network QoS system architecture, with QoS functionalities such as QoS provision, QoS control, and QoS management on each abstraction layer. We expect that this requires extensions to existing design languages such as UML or SDL. Finally, implementations are to be generated from design models. In our group, we have a complete development process and tool chain for model-driven development. It is a challenging task to extend them to QoS-aware system development.

Acknowledgments. The work presented in this paper was (partially) carried out in the BelAmI (Bilateral German-Hungarian Research Collaboration on Ambient Intelligence Systems) project, funded by German Federal Ministry of Education and Research (BMBF), Fraunhofer-Gesellschaft and the Ministry for Science, Education, Research and Culture (MWWFK) of Rheinland-Pfalz.

References

1. Schmitt, J.: Heterogeneous Network Quality of Service Systems. Kluwer Academic Publishers, Boston (2003) ISBN: 07937410X
2. Jin, J., Nahrstedt, K.: QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy. IEEE MultiMedia 11(3), 74–87 (2004)
3. Frølund, S., Koistinen, J.: QML: A Language for Quality of Service Specification. Technical Report HPL-98-10, pp. 63, Software Technology Laboratory, Hewlett-Packard Company (1998)
4. Aagedal, J.Ø.: Quality of Service Support in Development of Distributed Systems. PhD thesis, University of Oslo, Oslo, Norway (2001)
5. Vanegas, R., Zinky, J.A., Loyall, J.P., Karr, D., Schantz, R.E., Bakken, D.E.: QuO's Runtime Support for Quality of Service in Distributed Objects. In: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, UK, pp. 207–222 (1998)
6. Florissi, P.G.S.: QoSME: QoS Management Environment. PhD thesis, Columbia University (1996)
7. Campbell, A.T.: A Quality of Service Architecture. PhD thesis, Computing Department, Lancaster University (1996)
8. Koliver, C., Nahrstedt, K., Farines, J.M., Fraga, J.D.S., Sandri, S.A.: Specification, Mapping and Control for QoS Adaptation. Real.-Time. Systems 23(1-2), 143–174 (2002)

9. Huard, J.F., Lazar, A.A.: On QoS Mapping in Multimedia Networks. In: 21th IEEE Annual International Computer Software and Application Conference (COMPSAC'97), IEEE Computer Society Press, Los Alamitos (1997)
10. Fukuda, K., Wakamiya, N., Murata, M., Miyahara, H.: QoS Mapping between User's Preference and Bandwidth Control for Video Transport. In: 5th International Workshop on Quality of Service (IWQoS'97), Kluwer Academic Publishers, Dordrecht (1997)
11. DaSilva, L.A.: QoS Mapping Along the Protocol Stack: Discussion and Preliminary Results. In: Proceedings of IEEE International Conference on Communications (ICC'00). vol. 2. New Orleans, LA, pp. 713–717 (2000)
12. Webel, C., Fliege, I., Geraldy, A., Gotzhein, R., Krämer, M., Kuhn, T.: Cross-Layer Integration in Ad-Hoc Networks with Enhanced Best-Effort Quality-of-Service Guarantees. In: Proceedings of World Telecommunications Congress (WTC 2006), Budapest, Hungary (2006)