Generic Function Schema as a Means for Similar-Fashioned Operations on Heterogeneous Connection Properties

Mark Yampolskiy[1,4], Wolfgang Hommel[2,4], David Schmitz[2,4], Michael Schiffers[3,4]
myy@isis.vanderbilt.edu, hommel@lrz.de, schmitz@lrz.de, schiffer@nm.ifi.lmu.de

[1]*Vanderbilt University (VU),* [2]*Leibniz Supercomputing Centre (LRZ),* [3]*Ludwig Maximilians University Munich (LMU),*
[4]*Munich Network Management (MNM) Team*

***Abstract*—Graphs are often used to model interconnected topological objects with different connection properties. Path finding in a weighted graph belongs to the classical problems of graph theory. Whereas the addition of the edges' weights as an aggregation and the interpretation of a smaller resulting sum as the preferable path works very well in applications like path computations, e.g., for road maps, it is not always applicable to those connections in computer networks that need to fulfill multiple independent Quality of Service (QoS) criteria simultaneously. Until now, special solutions are implemented – often manually – for each new service and for each QoS parameter separately. As the development of novel customer-tailored network services often relies on different connection properties and their combinations, a generic treatment of QoS parameters becomes a critical factor for rapid development and network service rollout. In this article, we present a generic function schema for treating multiple independent QoS parameters in a similarly fashioned way. Our work fosters efficient routing algorithms that are considering multiple connection properties and corresponding constraints at the same time, as they are required, for example, in Future Internet infrastructures with end-to-end QoS guarantees and in dynamic survivability-aware environments.**

Keywords-graph theory; multi-weighted graphs; QoS; QoS aggregation; QoS comparison.

I. Introduction

Obviously, network connections are meanwhile broadly used as a basis for or as an integral part of the services that are realized upon them. Examples can be found in areas like Internet-telephony, video-conferencing and video-on-demand, connectivity for Grid cooperation, IT service outsourcing, etc. Common to all these examples is that the overall service quality directly depends on the combination of multiple *Quality of Service* (QoS) parameters of the underlying network connections. For instance, services like telephony, e.g., realized using standard VoIP (Voice over IP) protocols, are very sensitive to transmission jitter, as the human ear is very sensitive to the delay variation; on the other hand, video streaming depends primarily on the data rate, so that the end-users do not have to wait periodically for the transmission of the next portion of a high-resolution video; multi-player gaming in LANs and e-Sports over the Internet often demand low latency and are sensitive to packet or even connection loss; network connections for business and scientific applications often combine requirements for multiple QoS parameters at the same time and specify tighter thresholds than the usual consumer applications' demands.

This work was done while Mark Yampolskiy was employed by the LRZ and worked for the DFN in the Géant project.

The fulfillment of the service- or customer-specific *end-to-end* (E2E) requirements is only possible if all respective QoS parameters are considered during the path computation (routing), either as basis or as derived parameters . The state-of-the-art routing algorithms that are taking into account E2E requirements typically operate on weighted graphs in almost the same manner as approaches operating on multi-weighted graphs, i.e., on graphs with multiple weights associated with the single edge representing values of multiple independent QoS parameters.

In graph theory, it is common to use the *addition* of edge-weights as an aggregation function with the path with the smaller sum as the most preferred alternative. Such procedures are very well suited in applications like path computations for road maps in mobile navigation assistants. However, the approach is not always applicable to connections in computer networks with combined QoS parameters as the two network QoS parameters considered most often (bandwidth and delay) show. Whereas the typical parameter treatment is applicable to "delay", a different function is needed for "bandwidth": The aggregation function needs to choose the minimum bandwidth of all involved connection segments and larger values are preferred over smaller ones. In general, adequate QoS aggregation functions are significantly more complex than sum-of or minimum-of if other QoS parameters like reliability and availability have to be regarded as well.

Until now, there is no generally applicable solution to overcome these difficulties. On the other hand, however, the time for the development of new services with customer-specific QoS parameters is becoming a crucial success factor.

In order to cope with the high variety of customer- and service-specific requirements, we have presented in [1] a generic function schema which allows the treatment of various arbitrary network QoS parameters in a similar-fashioned way. The proposal in [1] includes an efficient way to distinguish between different QoS parameters; a standardized general treatment for the aggregation of and the comparison between values of a particular QoS parameter; and the support of customer-relevant combinations of arbitrary QoS parameters. The functions proposed in [1] can be used by routing procedures in order to find a path fulfilling the E2E constraints or as part of the monitoring of established connections in order to ensure the fulfillment of committed E2E connection qualities. However, [1] is limited to operations on properties that could be called *basis QoS parameters*. These are QoS parameters like bandwidth or delay, which can be measured directly. [1] is not applicable to *derived QoS parameters* that result from combining multiple basis QoS parameters with arbitrary formulae. A good example of such a parameter is “availability”, defined as the ratio of the total time a component is capable of being used during a given time interval to the length of this interval.

Consequently, while basis parameters induce metrics the dependencies of which on other variables is not explicit, derived parameters induce metrics where the influencing variables are explicitly considered. The important practical benefit of derived QoS parameters is their conceptual separation from basis parameters (and thus from dedicated measurements). For example, reliability (a derived QoS parameter) – defined as the probability that a service will perform its intended function during a specified period of time –, is not solely tied to uptime (a basis parameter). It could also be coupled to reaction times, response times or – in the context of a reliability-scalability metric – even to the number of users when defining it as the Mean Time to Failure (MTTF) in dependency of the number of users.

Despite these obvious advantages, however, the determination of derived QoS parameters induces a significant increase in complexity as the operations to aggregate the parameters are not necessarily homogeneous (summations may be combined with min-max-considerations and set theoretic intersections).

The remainder of the paper is organized as follows: We first analyze the state of the art in Section II. In Section III, we summarize first the main results of our original paper [1] which are necessary to understand the subsequent sections. We then generalize from the basis QoS parameters to derived ones and show how operations on them can be formally specified. In Section IV, we extend the discussion to properly treating the special problem of value ranges. In Section V we demonstrate the applicability of our approach to path finding problems in networks. Section VI discusses the operations required for aggregating derived QoS parameters, before we present an information model for derived QoS parameters in Section VII. Section VIII summarizes the resulting big picture and presents a concrete example. Finally, we conclude the paper in Section IX, where we also give an outlook to our ongoing work.

II. State of the Art

Most routing algorithms are based on graphs that have a single fixed value associated with their edges as weights. This representation is then used for finding a path (often the shortest path) between arbitrary nodes of the graph. However, such graphs do not reflect all specifics of computer networks (see Figure 1). For instance, different quality classes of the network infrastructure canonically lead to significant variances in parameter values. In order to process such value ranges, graphs can be transformed into so called *multigraphs* where nodes may be directly connected by one or more edges. Even in the simple case of weight ranges for a single property, such transformations can significantly increase the graph processing complexity. If multiple connection properties with value ranges have to be considered at the same time, the complexity increases even more drastically. Therefore, in [2], we proposed an information model which is able to describe value ranges. Consequently, such a description raises the necessity of adequate operations on ranges.

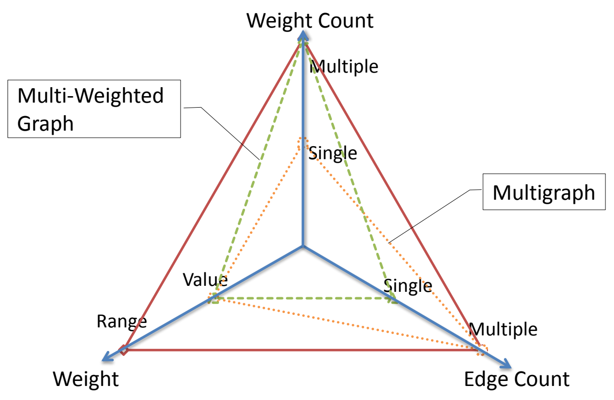

Figure 1. Classification of graph properties [1]

Graphs that support multiple weights at the same time are known as *multi-weighted graphs*. Such graphs are hardly investigated yet. In [7], a very good overview of the state of the art is given. It shows that path finding in multi-weighted graphs is in general an NP-complete problem. As path finding in multi-weighted graphs violates Bellman’s optimality principle [8], routing algorithms that require this principle, e.g., Dijkstra’s algorithm, cannot be used. Additionally, the handling of multiple properties at the same time is not solved adequately. Currently, the common understanding is to describe multiple properties as value vectors. This allows the use of vector-addition as property aggregation operation. For a comparison of weight vectors, the concept of *non-dominance* has been established [4]: A vector A is non-dominant to vector B only if all of its weight elements, i.e., property values, are smaller or equal to the corresponding elements of vector B.

As for single-weighted graphs, for multi-weighted graphs addition is also the pre-dominant aggregation function with a smaller value being the better one. Even if limitations of these operations w. r. t. the application to computer networks are long known, only workarounds have been proposed so far. For instance, in [9] the addition of $log(weight)$ is proposed if the true aggregation function for weights is multiplicative.

Alongside with the directly measurable QoS parameters, the necessity of parameters like service availability or reliability is very well agreed. Such quality parameters are defined in *IT Service Management* (ITSM) frameworks like ITIL [11] or e-TOM [12] and are widely used in *Service Level Agreements* (SLA) among network providers and carriers. Furthermore, more comprehensive research areas like survivability and dependability rely on derived QoS parameters (as, for example, pointed out in [10]) with arbitrary aggregation functions, corresponding weight comparison procedures, and improved handling of value ranges. While solutions to the first two aspects will be described in Section III, an efficient approach for handling value ranges during path finding will be presented in Section IV-A.

Besides these purely technical aspects, organizational specifics have to be considered as well. The so called policy-based routing between domains, does not only take technical aspects into account. Rather, it focuses on provider-specific interests. Along with very restrictive information and management policies, which are out of the scope of this paper, network operators and service providers are generally interested in the reduction of the resources that are required for a high-quality service delivery.

III. Operations on Connection Properties and their Generalization

In this section, we present a solution to function generalization regarding both single properties and property sets.

A. Functions for operations on a single property

During path finding the properties of the edges have to be aggregated. Typically, simple arithmetical addition is used as an aggregation function. As discussed in Section I, this is not necessarily the case for every QoS parameter. Furthermore, as discussed in [2], in the case of inter-domain connections each *Service Provider* (SP) may have access only to information regarding his own technical infrastructure which may not be sufficient to determine all relevant connection properties. In this case also the aggregation of the partial views of involved SPs at the same inter-domain connection is needed. The calculation of QoS properties of the inter-domain link from two partial views is not necessarily identical to the aggregation of two physical connections of the same type and length. For instance, when describing a connection with the property *delay*, not only the delay caused by the network cable should be considered, but also the delay caused by the active and passive network components used by each single SP; obviously, this varies between SPs.

If customer-specific end-to-end quality-of-service constraints need to be met, the value of the already found (partial) route has to be compared to these constraints during the path finding process. For path optimization it is also necessary to compare the values of several alternatives in order to choose the better one. In opposite to the case classically treated in graph theory, the meaning of what is *better* may vary between different QoS parameters. Regarding the examples mentioned above, for bandwidth a bigger value can be considered as a better one, however for delay a smaller value is the more preferred one.

Consequently, with each supported connection property operations for value aggregation and comparison have to be associated. As we will see, the necessary mathematical operations are not always as simple as adding values or selecting the minimum.

B. Associating operations with properties

In IT industry, new technologies and services are evolving very fast. Therefore prior to the association of operations with properties, a distinction between existing and projected properties is needed. We propose to assign a globally unique ID to each supported property. In order to ensure the global uniqueness of IDs, we propose to use a registration tree. Additionally to the distinction between properties, using a registration tree has another very important advantage. As multiple functions have to be associated with each supported property, it can be realized by the definition of the functions together with the registration of their property-ID (see Figure 2). Additionally, this will ensure the identity of functions used among SPs.

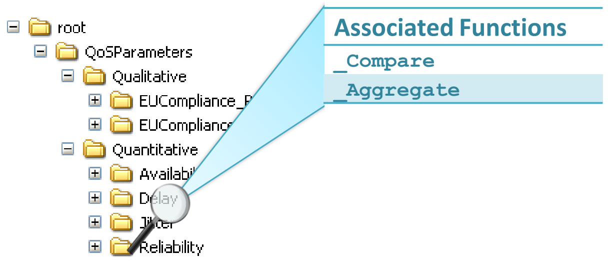

Figure 2. Registration tree example [1]

C. Comparison and aggregation of multiple properties

Based on the previous definition, we introduce an approach for the handling of m different properties with the globally unique IDs $\mathrm{ID}_1, \ldots, \mathrm{ID}_m$. In graph theory, it is common practice to use vectors in order to describe multiple weights associated with a single edge or a path in general. For any path in a graph with m properties, the weight can be specified as $\overrightarrow{U} ::= (u_1, \ldots, u_m) \in \mathbb{R}^m$. In this definition, u_j is the weight of the j$\underline{^{th}}$ property with ID$_j$. The order of properties in the weight vector can be arbitrary, as long as the placement of the properties is identical among all weight vectors. Further, for the edges of a path being enumerated from 1 to n, the weight of an edge with index i will be referred to as follows: $\overrightarrow{W}^i ::= (w_1^i, \ldots, w_m^i) \in \mathbb{R}^m$.

In order to calculate the weight vector $\overrightarrow{P}$ of the path consisting of n edges with weights $\overrightarrow{W}^1, \ldots, \overrightarrow{W}^n$, we first introduce an aggregation function for two weight vectors as follows:

$$\overrightarrow{\text{Aggr}}(\overrightarrow{U}, \overrightarrow{V}) ::= (\text{Aggr}_1(u_1, v_1), \ldots, \text{Aggr}_m(u_m, v_m))$$

This definition is based on m aggregation functions for each property. The aggregation functions Aggr$_i$ $(i = 1, \ldots, m)$ are associated with the property ID in the registration tree. We assume that all properties are independent of each other, i.e., they can vary without influencing the values of other properties. Furthermore, we assume that the binary operations defined by aggregation functions fulfill associative and commutative laws. Then we inductively define the computation of the whole path weight from weights of involved segments as follows:

$$\overrightarrow{\text{Aggr}}(\overrightarrow{W}^1, ..., \overrightarrow{W}^n) ::= \overrightarrow{\text{Aggr}}(\overrightarrow{\text{Aggr}}(\overrightarrow{W}^1, \overrightarrow{W}^2), ..., \overrightarrow{W}^n)$$

Similar to the aggregation, we define the comparison of property vectors based on the comparison between identical properties. Corresponding to the *non-dominance* concept described in [4] we define that vector $\overrightarrow{U}$ is better than $\overrightarrow{V}$ if and only if all properties in the first vector are better than the corresponding properties of the second vector. In order to denote that property u_i of vector $\overrightarrow{U}$ is better than the corresponding property v_i of vector $\overrightarrow{V}$, we use the symbol "$\prec$". In contrast to the comparison of single values, it is possible that some properties of the first vector are better and some others are worse than those of the second vector. This situation should be treated as “indefinite”. We depict this with the symbol "$\neq$". The comparison of two property sets can thus be defined as follows:

$$\overrightarrow{Compare}(\overrightarrow{U}, \overrightarrow{V}) ::= \begin{cases} =, \text{ if } & \forall 1 \le i \le m : u_i = v_i \\ \prec, \text{ if } & \forall 1 \le i \le m : (u_i \prec v_i \\ & \quad \vee\ u_i = v_i) \wedge \\ & \exists 1 \le j \le m : u_j \prec v_j \\ \succ, \text{ if } & \forall 1 \le i \le m : (u_i \succ v_i \\ & \quad \vee\ u_i = v_i) \wedge \\ & \exists 1 \le j \le m : u_j \succ v_j \\ \neq, \text{ if } & \exists 1 \le i \le m : u_i \prec v_i \wedge \\ & \exists 1 \le j \le m : u_j \succ v_j \end{cases}$$

IV. Treatment of Value Ranges

Some typical aspects of computer networks are not directly addressed by classical graph theory. In this section we propose the treatment of value ranges which can be associated with connection segments (graph edges) instead of multigraphs.

A. Path finding with value ranges

Physical network connections usually cannot be realized with a single property set because properties like bandwidth might vary in a wide range. A good example is the variation of achievable delays for a single logical connection, as it can be realized by different physical connections. Consequently, the property of the whole end-to-end (E2E) path between two endpoints may vary as well. We will refer to the value range of a particular path $path$ as

$$\overset{\rightleftharpoons}{W}^{path} = (\overrightarrow{W}^{path}_{min}, \overrightarrow{W}^{path}_{max}) \in \mathbb{R}^m \times \mathbb{R}^m,$$

i.e., the supported value range for the given path can vary from $\overrightarrow{W}^{path}_{\text{min}}$ to $\overrightarrow{W}^{path}_{\text{max}}$.

It is obvious that the path found between two endpoints can only be feasible if the best possible value fulfills the E2E constraints specified by customer (see Figure 3). Therefore, we propose to operate with the best values of the available connection segments during the path finding process.

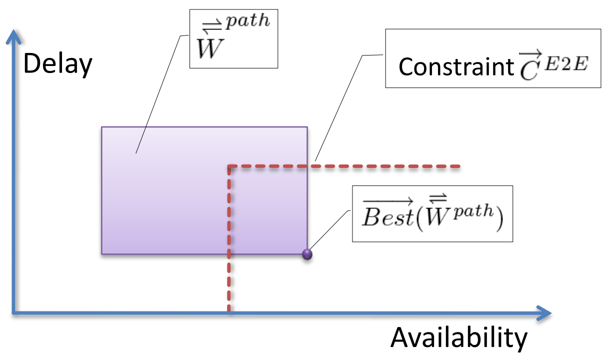
$\overset{\rightleftharpoons}{W}^{path}$
Delay
Constraint $\overrightarrow{C}^{E2E}$
$\overrightarrow{Best}(\overset{\rightleftharpoons}{W}^{path})$
Availability

Figure 3. Fulfillment of end-to-end constraints [1]

We assume that all simultaneously considered path properties can vary independent of each other. Under this assumption, we define the selection function $Best$ for the best possible value of a path as follows:

$$\overrightarrow{Best}(\overrightarrow{\overrightarrow{W}}^{path}) = \overrightarrow{Best}(\overrightarrow{W}^{path}_{min}, \overrightarrow{W}^{path}_{max})$$

$$\overrightarrow{Best}(\overrightarrow{U}, \overrightarrow{V}) ::= (Best_1(u_1, v_1), \dots, Best_m(u_m, v_m))$$

$$Best_i(u_i, v_i) ::= \begin{cases} u_i, \text{ if } u_i \prec v_i \\ v_i, \text{ otherwise} \\ \text{for } 1 \le i \le m \end{cases}$$

Please note that this definition is applicable not only to a path as a whole but also to any path segment.

B. Considering service provider interests: Optimization of resource usage

In contrast to customers, the service providers are usually interested in a reduction of resources used for service realization. This means that the requested service quality should not be the best possible one, but rather the one closest to the customer constraints. For paths complying with the E2E constraints, i.e., $\overrightarrow{Best}(\overrightarrow{\overrightarrow{W}}^{path}) \prec \overrightarrow{C}^{E2E}$, we distinguish between three cases as depicted in Figure 4, given the weights of alternative paths A, B and C:

- All worst properties of the considered path are worse than the constraints (see ”Path A”)
- All worst properties of the path are better than the constraints (see ”Path B”)
- The worst properties of the path are for some properties worse and for other properties better than the constraints (see ”Path C”)

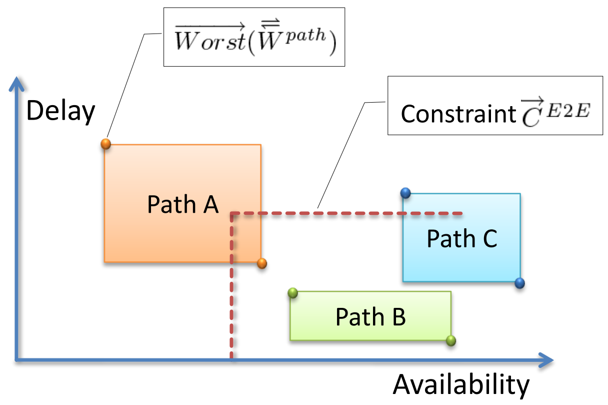

Figure 4. Pathweights of paths complying to constraints [1]

In order to distinguish between these alternatives, the function $Worst$ for the selection of the worst possible value of the found path can be defined as the opposite to $Best$.

In the case equivalent to ”Path B”, the worst possible value can be requested during the link ordering process. In the two remaining cases, an approximation to the constraint value should be performed. As the properties are independent of each other, such an approximation can be done separately (or even in parallel) for each affected property.

The whole E2E path weight is the aggregation of the weights of the involved parts. A possible gradation between the maximum and minimum values of connection parts is depicted in Figure 5. The E2E approximation of the path weight for a single property can be done in different ways. It can be seen as a knapsack-like problem with an intention to find a fit most close to the E2E constraint. We argue against this approach, as it may prevent the on-demand adaptation of requested service parts parameters. Instead we favor a ”fair split” among all connection parts. For each property i, we propose to use a divide-and-conquer strategy as follows:

1) For each connection part j with a value range between $w^j_{i,min}$ and $w^j_{i,max}$ we compute values $w^j_{i,best}$=Best$_i$ ($w^j_{i,min}$, $w^j_{i,max}$) and $w^j_{i,worst}$=Worst$_i$($w^j_{i,min}$, $w^j_{i,max}$).
2) For each connection part j we compute the realizable value $\left\lfloor \frac{w^j_{i,best}+w^j_{i,worst}}{2} \right\rfloor$.
3) If the computed path value $\sum_{j=1}^{k} \left\lfloor \frac{w^j_{i,best}+w^j_{i,worst}}{2} \right\rfloor$ is equivalent to the E2E constraint for the selected property, the selected values can be used as a result of this optimization.
4) If the computed path value is better than the E2E constraint, the computed values for connection parts should be used in the next step as $w^j_{i,best}$, otherwise as $w^j_{i,worst}$.
5) We propose to limit the number of optimization steps. If the number of maximal optimization steps is reached, the latest $w^j_{i,best}$ for each connection part should be used as an approximation value. If the amount of the maximum optimization steps is not reached yet, this procedure shall be repeated beginning with step (2).

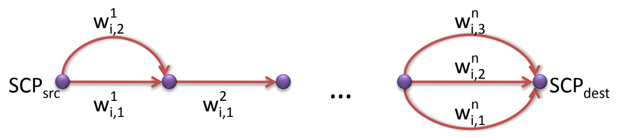

Figure 5. Possible gradation of values for different path segments for property i [1]

Please note that in order to reflect the ”better/worse” comparison instead of ”smaller/bigger” one, we define the unary operator ”$\lfloor\ \rfloor$” as follows: the result should be the worst realizable value which is equal or better than the value enclosed in the brackets.

V. Application to Search Problems

In Figure 6, we present a path finding algorithm, which illustrates the usage of our operators. In the pseudo-code, a Deep First Search (DFS) strategy is used for finding a path complying with multiple QoS constraints $\overrightarrow{C}^{E2E}$.

```
**MCP** (*nodeCur*, *nodeDest*, $\overrightarrow{W}^{path2cur}$, $\overrightarrow{C}^{E2E}$)

  **if** (nodeCur == nodeDest)
    BacktracePath (*nodeCur*);
    return TRUE;
  **end if**

  MarkNode (*nodeCur*);

  **for** each neighbor *nodeNbr* of *nodeCur*
    **if** (*not* Marked (*nodeNbr*))
      $\overrightarrow{W}^{path2nbr}$ = $\overrightarrow{Aggr}$ ($\overrightarrow{W}^{path2cur}$, $\overrightarrow{Best}(\overset{\rightrightarrows}{W}^{cur2nbr})$)

      **if** ($\overrightarrow{W}^{path2nbr} \prec \overrightarrow{C}^{E2E}$)
        **if** (MCP(*nodeNbr*, *nodeDest*, $\overrightarrow{W}^{path2nbr}$, $\overrightarrow{C}^{E2E}$))
          BacktracePath (*nodeCur*);
          return TRUE;
        **end if**
      **end if**
    **end if**
  **end for**

  UnmarkNode (nodeCur);
  **return** FALSE;
```

Figure 6. Use of the new operators in a path finding algorithm [1]

The presented algorithm solves the so-called *multi constrained path finding* (MCP) problem. The function requires four parameters. The first two parameters ($nodeCur$ and $nodeDest$) specify nodes in the graph, between which a path has to be found. As the MCP function is called recursively, the $nodeCur$ specifies the end of the intermediately considered path. The weight of the intermediate path is given in the third parameter $\overrightarrow{W}^{path2cur}$. Finally, $\overrightarrow{C}^{E2E}$ are always the E2E-constraints between two endpoints.

The function first checks whether the destination node is reached yet. If it is the case, the $BacktracePath$ function is called in order to memorize the node in the path between two endpoints. Then the value $TRUE$ is returned which signals that a path with acceptable properties has been found.

If the end node is not yet reached, the $nodeCur$ is marked using the function $MarkNode$. This is a common practice in DFS-algorithms in order to prevent loops. In the following $for\ each$ loop all neighbors of $nodeCur$ are considered that have not been marked. For each neighbor $nodeNbr$ a weight $\overrightarrow{W}^{path2nbr}$ of an path between start and $nodeNbr$ nodes is computed. Corresponding to Section IV-A, the best possible value of the considered segment weight $\overset{\rightrightarrows}{W}^{cur2nbr}$ is aggregated with the intermediate sum $\overrightarrow{W}^{path2cur}$. If the computed weight of the new intermediate path is still better than E2E-constraint $\overrightarrow{C}^{E2E}$, the MCP function is called recursively. This time, $nodeNbr$ is used to mark the end of the intermediate path. If the function returns TRUE, the node is saved in order to backtrace the path; subsequently TRUE is returned. If the call to the MCP function was not successful, the next neighbor has to be considered likewise. If all neighbors have been considered without any success, the node $nodeCur$ is unmarked and the value $FALSE$ is returned.

Please note that for the sake of simplicity in this algorithm at most one connection between two nodes is supported. An extension for multigraphs would require an additional loop for all edges between two interconnected nodes. Furthermore, also the backtracking function should be extended in this case, in order to track not only nodes along the path, but also along used edges.

VI. Aggregation of Derived QoS Parameters

The comparison of the derived QoS parameters does not differ from the one of basis QoS parameters. The reason is that the derived QoS parameters are commonly defined as values that have to be compared with the defined thresholds. However, the aggregation of the values of derived QoS parameters is significantly more complex.

In the organizational domain of a single service provider, all basis QoS parameters required for the computation of derived QoS parameters are available. Therefore the derived QoS parameter – from the perspective of this provider – can be calculated end-to-end. Consequently, there is no necessity to aggregate values of derived QoS parameters. However, this is not applicable to multi-domain network connections. Figure 7 presents an end-to-end connection crossing organizational boundaries of three providers and consisting of five connection segments.

The complexity of the aggregation can be demonstrated using the QoS parameter *Availability* typically found in *Service Level Agreements* (SLA). This QoS parameter is computed as *up-time*, i.e., the time during which the service was fully operational, divided by *total-time*, i.e., the time during which the availability should be measured. If in the example depicted in Figure 7 all connection segments are unavailable for one hour, the availability of each single segment is 95.83%. However, the availability of the whole end-to-end connection cannot be exactly computed based on these values. Instead, the up-time of the whole connection needs to be calculated first. If we abstain from the consideration of the possible time deviations among different domains, the total time of the whole connection need not have to

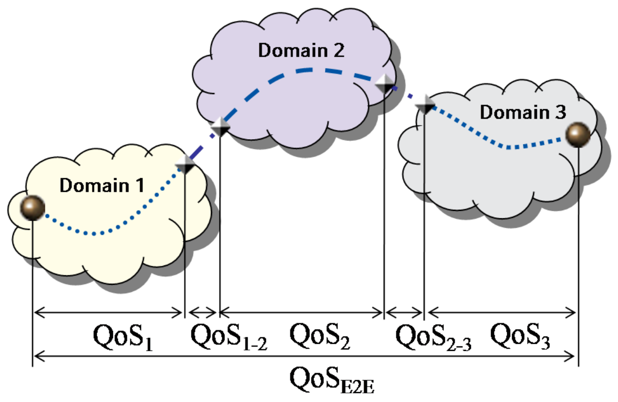

Figure 7. Composition of E2E connection quality [2]

be computed, instead it can be used as defined in the SLA. Returning to up-time calculation, this is again a derived QoS parameter, which can only be computed as the length of the intersection of up-times in all involved segments.

Generalizing our previous discussion, the values of derived QoS parameters can be seen as a projection from an m-dimensional vector of underlying (basis or derived) QoS parameters to the single value of the derived QoS parameter:

$$\text{Derive}_{QoS_{derived}} : QoS^m_{base} \rightarrow QoS_{derived}$$

Please note that the values of QoS parameters have not necesserily be the real numbers, e.g., up-time periods can be described as an conjunction of contiguous up-time periods, which in turn can be described based on the start and end time of this period. Further, the projection from the source to the target spece depends on the QoS_{dest}, i.e., the exact formula how the derived QoS value is calculated from the underlying QoS parameters. Furthermore, the destination QoS parameter is not necessarily one of the source QoS parameters, but we do not restrict this case.

In order to aggregate derived QoS values of two connection segments, we first have to calculate the aggregate of the underlying QoS values. This means that we have to access the values used in the $Derive$ projection. We call this access to underlying values $Base$:

$$\text{Base}_{QoS_{derived}} : QoS_{derived} \rightarrow QoS^m_{base}$$

Please note that the $Base$ operation cannot be realized on values alone, as the $Derive$ projection applied on various inputs in the source space can produce identical results in the target space. For instance, the mentioned availability of 95.83% can be achieved regardless at which time the service was unavailable for 1 hour; moreover, the unavailability time should not be a single continuous time period.

The necessity for the Base operation raises the requirements on the information model, which is used for the representation of the values of the various QoS parameters. In the case of derived QoS parameters not only the derived value itself, but also the underlying values used for their computations should be represented. Furthermore, such *derived/base*-relations should be recursive, with the leafs of the QoS-derivation-tree representing only basis QoS parameters.

Based on our previous discussion, we refine the function for the aggregation of two values belonging to derived QoS as follows:

$$Aggr(q,v) ::= \begin{cases} Aggr^{basis}(q,v), & \text{if } q,v \text{ are basis QoS} \\ Aggr^{derived}(q,v), & \text{if } q,v \text{ are derived QoS} \end{cases}$$

$$Aggr^{derived}(q,v) ::= Derive(\overrightarrow{Aggr}(Base(p), Base(q)))$$

where $Base$ and $Derive$ operations are specific to the QoS parameter of q, v, and the result of their aggregation. The vector aggregation $\overrightarrow{Aggr}$ is defined in Section III-C.

Please note that in opposite to our previous discussion these are not precise mathematical formulas that can immediately be applied on values. Instead these pseudo-formulas specify the order of computations, or the order and encapsulation of function calls as they are understood in computer science. For aggregation of two values of the same QoS type one has to distinguish first whether these QoS parameters are basis ones or derived ones. If they are basis ones, the aggregation of the values can be done according to the mathematical formula specified for the particular QoS in the registration tree. If they are derived ones, a more complex computation of the aggregated value is required. This include that the basis values of the derived QoS parameter should be accessed first. As the $Base$ operation can result in one or more underlying QoS parameters, the aggregation of two vectors is required. After aggregation of the underlying QoS values the values of the resulting vector have to be used in the formula for the computation of the derived QoS parameter.

Please note that following the $Base$ operation, a vector aggregation was used. This means that according to the definition of vector aggregations in Section III, this results in the aggregation of vector elements, which are values of either basis or derived QoS parameters. Consequently, again either basis or derived aggregation rules have to be applied, which may result in further $Base$ operations, until the aggregation on the values of the underlying basis QoS parameters can be performed. This can be seen as the navigation in the QoS parameter *derivation tree*.

Please note further that the distinction between aggregation of basis and derived QoS parameters determines how the aggregated value have to be computed. Regardless of the QoS class, the aggregation function has to be associated with its ID as proposed in Section III.

VII. Information Model supporting Basis and Derived QoS Parameters

As discussed above, the distinction between different QoS parameters can be realized based on their globally unique IDs. In [2], we have introduced an advanced information model, which can be used to describe available network connection segments as well as their properties, which includes various combinations of qualitative and quantitative QoS parameters as well as of the available management functionality (see Figure 8). However, the information model in its original form is not sufficient for the description of derived QoS parameters, as the computation on them, as defined in Section VI, requires the knowledge of the basis properties, from which the particular QoS parameter is derived.

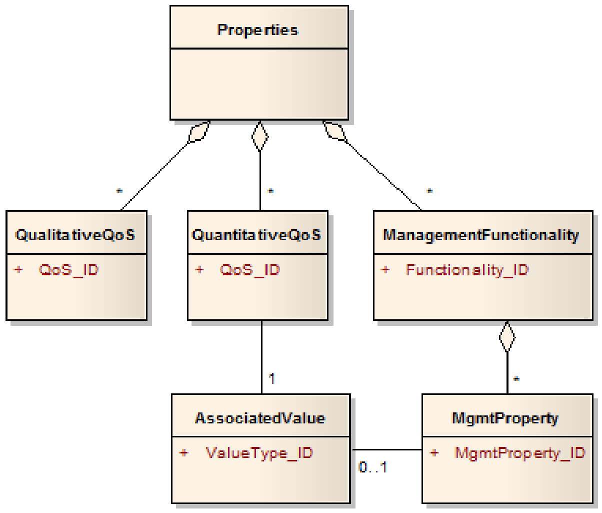

Figure 8. Connection segment properties [2]

In order to support the derivation of QoS parameters from each other, we extend this model with an additional view (see Figure 9). We define that every QoS parameter is derived from a basis class PROPERTY. The only purpose of this class is to show that every property can be either stand-alone, i.e., basis QoS, or derived from one or more other properties, i.e., derived QoS. The relation between derived and underlying QoS parameters is specified as a reflexive aggregation of the PROPERTY class. This aggregation called DerivedFrom has the cardinality $*$, which means that the same class can be used for the description of both basis and derived QoS parameters. Further, this aggregation involves the important operations $Base$ and $Derive$, as they have been defined in Section VI.

Please note that the proposed information model is highly extensible. For instance, it can be easily extended to support properties of energy consumption which are relevant, e.g., for energy efficient routing. The aggregation and comparison rules for such parameters can be defined as we have already

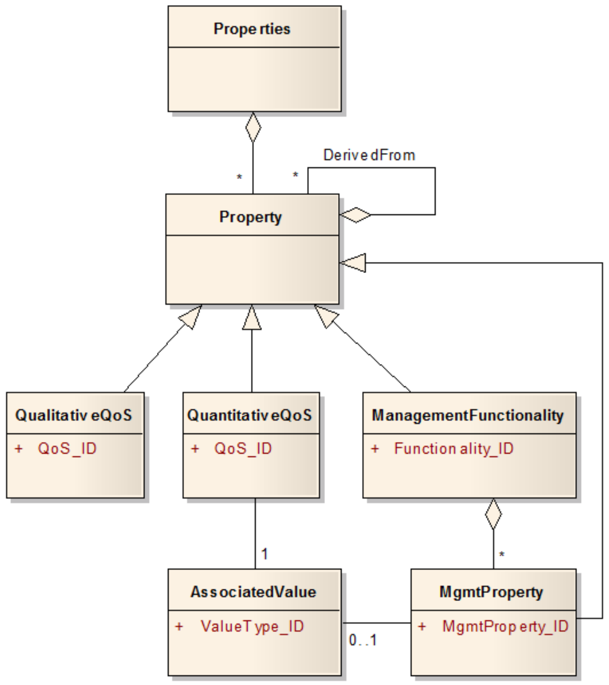

Figure 9. Deriving properties from each other

described for the QoS parameters. As the focus of the particular article lies on the function model, we abstain from the further discussion of the information model.

VIII. Putting it all together in an example

In order to illustrate the practicability of the proposed solution we refer to the situation of an E2E connection consisting of multiple segments (see Figure 7). As mentioned above, the aggregation is always needed during the routing process; it might also be needed for the monitoring of established connections. The latter is only necessary if no end-to-end measurements are possible and only connection segments can be monitored instead, which, however, is a very common situation in multi-domain connections. Further we only consider the situation that properties of different connection segments have to be aggregated together.

For the sake of simplicity, we say that only two QoS parameters (one basis and one derived QoS) are relevant for the connection: bandwidth and availability. The information about the connection segment's properties has to be structured according to the information model we have defined in Section VII. Currently, it is common that the components of distributed architectures communicate with each other via *web services*. This includes that the communication artifacts are encoded in an XML format. This is fully sufficient for the illustration purpose, as the XML format is human readable.

An XML schema can be derived directly from the defined information model. The choice of XML has an additional advantage, as it enables the description of service- or customer-requirements-adjusted combinations of supported connection properties. Please note that we intentionally avoid an explicit definition of the XML schema, as different realizations might be optimized for different purposed, e.g., for better parser performance or for human-readability.

One possible XML representation of the bandwidth and availability values of a single connection segment is depicted in Figure 10. In the XML code, the $Properties$ element encloses all properties of a segment, which are in this particular case two $QuantitativeQoS$ elements of the mentioned QoS parameters that are identified through their QoS_ID. Bandwidth belongs to the basis QoS class and is therefore structured very simple; it contains only the value and its metrics. The availability belongs to the class of derived QoS parameters. Therefore, for this parameter also basis values should be provided, which have been used for the calculation of the availability, i.e., in this case properties with IDs $UpTime$ and $TotalTime$. Please, note that also for these values properties are provided, from which they are derived.

```xml
<Properties>
  <QuantitativeQoS QoS_ID="Bandwidth">
    <AssociatedValue ValueType_ID="SingleValue">
      <SingleValue Value="1" Metric="Gbps"/>
    </AssociatedValue>
  </QuantitativeQoS>

  <QuantitativeQoS QoS_ID="Availability">
    <AssociatedValue ValueType_ID="SingleValue">
      <SingleValue Value="95.83" Metric="%"/>
    </AssociatedValue>
    <DerivedFrom>
      <QuantitativeQoS QoS_ID="UpTime">
        <AssociatedValue ValueType_ID="SingleValue">
          <SingleValue Value="23" Metric="Hrs"/>
        </AssociatedValue>
        <DerivedFrom>
          <AssociatedValue ValueType_ID="TimeRange">
            <TimeRange Begin="00:00" End="23:00" Metric="GMT"/>
          </AssociatedValue>
        </DerivedFrom>
      </QuantitativeQoS>
      <QuantitativeQoS QoS_ID="TotalTime">
        <AssociatedValue ValueType_ID="SingleValue">
          <SingleValue Value="23" Metric="Hrs"/>
        </AssociatedValue>
        <DerivedFrom>
          <AssociatedValue ValueType_ID="TimeRange">
            <TimeRange Begin="00:00" End="24:59" Metric="GMT"/>
          </AssociatedValue>
        </DerivedFrom>
      </QuantitativeQoS>
    </DerivedFrom>
  </QuantitativeQoS>
</Properties>
```

Figure 10. Example, basis and derived QoS parameters in XML

As discussed above, for the aggregation of values belonging to different connection properties, the property-relevant aggregation function should be used. According to our proposal, the definition of such functions should be associated with the property IDs in the registration tree. We assume that the QoS_IDs specified in XML file are sufficient for the unambiguous identification of QoS parameters and for access to the associated functions. Please note that the realization will require URNs with structure reflecting paths in the registration tree. Such functions can be defined, e.g., as the set of computation rules which should be executed or as a module of some interpreting programming language, which can be applied on demand.

For presentation purposes only, we define the aggregation functions for the example's QoS parameters as a pseudo-code (see Figure 11). All functions presented in the figure should be accessed from different locations in the registration tree. The aggregation rule for the bandwidth is very simple – it returns the smaller of to the two values. The rule for the calculation of the availability is more complex and follows the three steps described above. First, the basis values are obtained, which have been used for the calculation of the values p and q. Second, the aggregation function is executed on these basis values. Please note that the used aggregation functions correspond to the QoS parameters they are executed upon. Third, based on the results of the calculation in the previous step, the combined availability of two segments is calculated.

```
// Agggregation function for basis QoS "Bandwidth"
Bandwidth_Aggregate (p, q)
{
      if (p < q) return p;
      return q;
}

// Agggregation function for derived QoS "Availability"
Availability_Aggregate (p, q)
{
      // 1. Basis-operation for values
      //    Get basis values/value-vectors of derived QoS parameters
      p_base = Availability_Base (p);
      q_base = Availability_Base (q);

      // 2. Vector-aggregation of underlying QoS parameters
      //    Use for this purpose function defined for the basis QoS parameters
      //    In this case QoS parameters are time interval durations
      UpTime    = TimeDuration_Aggregate(p_base.UpTimeLength, p_base.UpTimeLength);
      TotalTime = TimeDuration_Aggregate(p_base.TotalTimeLength, p_base.TotalTimeLength);

      // 3. Derive-projection to the derived QoS "availability"
      Availability = UpTime / TotalTime;

      return Availability;
}

// Agggregation function for derived QoS "TimeDuration"
TimeDuration_Aggregate (p, q)
{
      // 1. Basis-operation for values
      //    Get basis values/value-vectors of derived QoS parameters
      //    In this case these are lists of all time intervals
      p_base = TimeDuration_Base (p);
      q_base = TimeDuration_Base (q);

      // 2. Vector-aggregation of underlying QoS parameters
      //    Use for this purpose function defined for the basis QoS parameters
      //    In this case QoS parameters are lists of time intervals
      TimeIntervalIntersection =
            TimeIntervalList_Aggregate (p_base.TimeIntervals, q_base.TimeIntervals);

      // 3. Derive-projection to the derived QoS "TimeDuration"
      JointTimeIntervalDuration = 0;
      for all TimeInterval in TimeIntervalIntersection
            JointTimeIntervalDuration += TimeInterval.IntervalDuration;

      return JointTimeIntervalDuration;
}

// Agggregation function for basis QoS "TimeInterval"
TimeIntervalList_Aggregate (p, q)
{
      // compute intersection of the all intervals in the interval lists "p" and "q"
      return IntervalListIntersection (p, q);
}
```

Figure 11. Pseudo-code for aggregation functions

The access to and execution of the comparison function is similar to the aggregation one. Therefore, we omit its explicit treatment. Instead we would like to discuss the disadvantage of the computational procedure we have defined for the aggregation of derived QoS parameters. Even though the defined procedure leads to the desired results, it requires $Base$ and $Derive$ operations every time values of two segments have to be aggregated. This means that for a connection with n segments, such operations should be done $2 * (n - 1)$ times ($n - 1$ pairs of 2 segments every time). We see a big potential for the improvement of this situation through the use of dynamic programming in the definition of aggregation functions. However, the exact optimizations have to be analyzed case by case for each supported derived QoS parameter.

IX. Conclusion and Future Work

In this article, we have defined a novel schema for the generic treatment of network connection properties. In order to support operations on arbitrary properties of network connections, we propose to associate five functions with the ID of each supported property. These functions are summarized in Table I. Three of these functions, which are used for property aggregation and comparison, are mandatory. The mandatory function _AGGREGATE_LINKPART is dedicated to compute the property of connection based on only partial views at the same inter-domain connection. For elaborated discussion about its necessity we refer to [2]. The remaining selection functions aim to simplify handling with value ranges. These functions are not mandatory, as they can be easily derived based on comparison function. Additionally to the mentioned five functions, for derived QoS parameters two operations $Base$ and $Derive$ should be defined. They can be either defined implicitly as a part of mentioned aggregation functions or explicitly as functions associated with the QoS ID in the registration tree.

Function class	Purpose
_COMPARE	Compare two values *a* and *b*. Result can be: "*a* is better", "*a* is worse", "*a* and *b* are equivalent"
_SELECT_BEST	Optional function returning the best value of a given value set
_SELECT_WORST	Optional function returning the worst value of a given value set
_AGGREGATE_LINKS	Aggregate property values of two links or paths
_AGGREGATE_LINKPARTS	Aggregate two partial views at the same link to a single link weight

Table I
FUNCTIONS FOR OPERATIONS ON A SINGLE QOS PARAMETER

Together with [2] and [3], which present an information model and a multi-domain routing procedure, the solution presented here is an integral part of our ongoing work enabling user-tailored connection services with guaranteed E2E quality. However, the generic operation handling proposed in this article is not restricted to the problem space described here. It can be used in alternative routing algorithms that are considering multiple properties, such as [5] and [6].

The presented proposal leaves some aspects unsolved, they will be addressed in further research as follows: In the first place, a meta-language for the description of property-related functions has to be selected; also, a concrete structure for the registration tree has to be proposed. In order to achieve this, a profound evaluation of alternatives is needed. In the case that a single global registration tree has to be used by multiple organizations, like it is the case for the internet registration tree, the description of equivalence relationships between different entries has to be addressed. Furthermore, the quality parameters of different network layers as well as user-faced services depend on the quality of the underlying layers they are realized upon. Therefore, a general description of such interdependencies and parameter transformations is essential in order to offer customer-demanded quality based on network-specific information.

Additionally, we plan to investigate the applicability of our approach on the services with other composition structures. For instance, the quantification of survivability capabilities requires conditional computations [13] leaving the strictly sequential compositions in favor of more flexible procedures including branches, loops, and cases.

Acknowledgment

The authors wish to thank the members of the Munich Network Management Team (MNM Team) [14] for fruitful discussions and valuable comments on previous versions of this paper. The MNM Team directed by Prof. Dr. Dieter Kranzlmüller and Prof. Dr. Heinz-Gerd Hegering is a group of researchers at Ludwig-Maximilians-Universität München, Technische Universität München, the University of the Federal Armed Forces and the Leibniz Supercomputing Centre of the Bavarian Academy of Science.

References

[1] M. Yampolskiy, W. Hommel, D. Schmitz, and M. K. Hamm, *Generic Function Schema for Operations on Multiple Network QoS Parameters*, Proceedings of The Second International Conference on Advanced Service Computing (SERVICE COMPUTAION 2010), pp. 126–131. Valencia, 2010.

[2] M. Yampolskiy, W. Hommel, P. Marcu, and M. K. Hamm, *An information model for the provisioning of network connections enabling customer-specific End-to-End QoS guarantees*, Proceedings of 7th IFIP/IEEE International Conference on Services Computing (SCC 2010), pp. 138–145. Miami, 2010.

[3] M. Yampolskiy, W. Hommel, B. Lichtinger, W. Fritz, and M. K. Hamm, *Multi-Domain End-to-End (E2E) Routing with multiple QoS Parameters. Considering Real World User Requirements and Service Provider Constraints*, Proceedings of The Second International Conference on Evolving Internet (INTERNET 2010), pp. 9–18. Valencia, 2010.

[4] F. A. Kuipers, *Quality of service routing in the internet: Theory, complexity and algorithms*, PhD thesis. Delft University Press, 2004.

[5] T. Korkmaz and M. Krunz, *Multi-constrained optimal path selection*, Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001), pp. 834–843. 2001.

[6] P. Van Mieghem, H. De Neve, and F. A. Kuipers, *Hop-by-hop quality of service routing*, Computer Networks, pp. 407–423. Elsevier, 2001.

[7] M. Ziegelmann, *Constrained Shortest Paths and Related Problems*, PhD thesis. VDM, 2007.

[8] R. Bellman, *The theory of dynamic programming*, Proceedings of the National Academy of Sciences of the United States of America, pp. 716–719. 1952.

[9] G. Bertrand, S. Lahoud, M. Molnar, and G. Texier, *Inter-Domain Path Computation with Multiple Constraints*. 2008.

[10] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, VOL. 1, NO. 1, Jan-Mar 2004, pp. 11-33

[11] V. Lloyd, C. Rudd, ITIL Service Design. The Stationery Office, 2007, ISBN 9780113310470

[12] TMForum, SLA Management Handbook, Release 2.5, GB917, 2005

[13] M. Schiffers, D. Kranzlmüller, *Folded Interaction Systems and their Application to the Survivability Analysis of Unbounded Systems*, Proceedings of 33th International Conference on Information Technology Interfaces (ITI 2011), pp. 97–102, Dubrovnik, Croatia, 2011

[14] *Munich Network Management Team (MNM Team) Homepage*, [Online: http://www.mnm-team.org], August 2010.