NQSL - A Specification Language for Network Quality of Service

– Technical Report 368/07 –

Christian Webel

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

Abstract. The provision of *Network Quality-of-Service (QoS)* is a major challenge in the design of future communication systems. Before designing and implementing communication systems, the network QoS requirements have to be specified. In this paper, we present a formal description technique for network QoS called *NQSL*, which is based on a formalization of network QoS requirements, including QoS mappings for cross-layer design and scalability. 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.

1 Introduction

One of the major challenges in wireless (ad-hoc) networks is the provision of network Quality-of-Service (network QoS), *i.e.* the quality of a service provided by the underlying communication system. The need for network QoS arises from the fact that, for state-of-the-art distributed user applications, it is essential, to offer their functionality with a certain degree of quality, which requires suitable communication mechanisms. State-of-the-art wireless distributed communication systems must offer proactive and intelligent behaviour in order to cope with varying channel quality and connectivity. In other words, wireless communication systems must facilitate changes at runtime, and these changes are to be performed according to an effective reasoning about user, environment, and system context. The realization of such adaptive behaviour can in fact be seen as one of the technological key challenges in the development of wireless communication systems supporting network Quality-of-Service.

One of the main drivers of adaptive behaviour is the need to maintain specific non-functional properties, *i.e.* a specific level of QoS for the provided services. Especially in the wireless domain, where resources (like bandwidth, energy, processing power, and memory) are inherently scarce and subject to frequent change, the systems need to manage their resources in a QoS-aware way. To this end, to

form a basis for corresponding adaptation mechanisms to work on, a major prerequisite is an explicit *specification* of QoS offers and correlated needs. Moreover, QoS as an inherently cross-cutting concern has to be considered from end-to-end and from user layer down to the hardware layer. The QoS requirement specifications hence reside on different layers of abstraction and need to be mapped on each other.

In previous work [1], we have introduced a formalization of network QoS. In particular, we have formalized the notions of QoS domain, QoS scalability, QoS mapping, and QoS specification to reach the aforementioned goals. In this paper, we build on and extend these results by providing a formal description technique called *NQSL*, the *Network QoS Specification Language*, to intuitivly specify the requirements on network QoS.

The remaining part of this paper is organized as follows: In Section 2, we summarize our formalization and specification of network QoS requirements from a previous paper. Section 3 introduces NQSL, the Network QoS Specification Language. Conclusions are drawn in Section 4.

2 Formalization of Network Quality-of-Service

In previous work [1], we have introduced a formalization, specification and formal mappings of network quality of service. In this section we briefly sum up the achieved results.

2.1 Formalization and Specification 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 need a *precise* and *well-defined* description of QoS. Formalization of network QoS is done by firstly identifying the *QoS domain*, and secondly by describing the *QoS Scalability*.

The *QoS domain* Q captures the QoS characteristics of a class of data flows, i.e. performance, reliability, and guarantee and is therefore 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. An element $q = (p, r, g)$ of Q is called *QoS value*. *QoS performance* describes efficiency aspects characterizing the required amount of resources and the timeliness of the service. The therefore identified QoS parameters are included in the *QoS performance domain* P with $P = P_1 \times \ldots \times P_n = \prod_{i=1}^{n} P_i$, where $P_1, \ldots, P_n$ are performance subdomains. The *QoS reliability* describes the *safety-of-operation* aspects characterizing the fault behaviour (e.g., loss rate and distribution, corruption rate and distribution, error

burstiness) and is defined as $R = \mathit{Loss} \times \mathit{Period} \times \mathit{Burstiness} \times \mathit{Corruption}$, with $\mathit{Loss} = \mathbb{N}_0$, $\mathit{Period} = \mathbb{R}_+$, $\mathit{Burstiness} = \mathbb{R}_+$, and $\mathit{Corruption} = \{r \in \mathbb{R} \mid 0 \leq r < 100\}$. The *QoS guarantee* describes the degree of commitment characterizing the binding character of the service. Four degrees of commitment are distinguished: *Best-effort*, *Deterministic*, *Statistical*, and *Enhanced Best-Effort*. QoS guarantee is formalized by the *QoS guarantee* domain as $G = \mathit{DoC} \times \mathit{Stat} \times \mathit{Prio}$, where $\mathit{Stat} = \{p \in \mathbb{R} \mid 0 < p \leq 1\}$, $\mathit{Prio} = \mathbb{N}$, and $\mathit{DoC} = \{\mathit{bestEffort}, \mathit{enhancedBestEffort}, \mathit{statistical}, \mathit{deterministic}\}$.

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. The domain of QoS scalability S is defined as $S = \mathit{Util} \times \mathit{Cost} \times \mathit{Up} \times \mathit{Down}$, where $\mathit{Util} = \{u \mid u : Q \rightarrow [0,1]\}$, $\mathit{Cost} = \{c \mid c : Q \rightarrow \mathbb{R}_+\}$, and $\mathit{Up}, \mathit{Down} \in \{x \in \mathbb{R}_+ \mid 0 \leq x \leq 1\}$. The elements of *Util* and *Cost* are called *utility functions* and *cost functions*, respectively. A utility function determines the usefulness of QoS values, a cost function c expresses the amount of needed resources, associating higher costs with scarcer resources. QoS values with the same utility (cost) ($\sim_{u(c)}$) are assigned to the same so-called *u(c)*-equivalence class of Q: $[x]_{u(c)} = \{q \in Q \mid q \sim_{u(c)} x\}$.

A *QoS requirements specification qosReq* defines the set of valid QoS values and a QoS scalability value and 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 a set $Q' \subseteq Q$ of valid QoS values. To obtain this Q', a preorder $\lesssim_u$ induced by the utility function is applied: $Q' = \{q \in Q \mid q_{min} \lesssim_u q \lesssim_u q_{opt}\}$.

We can stepwise reduce the QoS domain Q. Therefore, 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, $Q^u = \{q_1, \ldots, q_m\} \cap Q', \quad q_i = q \in [x]^i_u \mid \forall y \in [x]^i_u \,.\, q \lesssim_c y, \quad 1 \leq i \leq m$. A further reduction induces a derived QoS domain $Q^{u,c}$, discarding QoS values with higher cost, but less utility, $Q^{u,c} = \{q \in Q^u \mid \forall y \in Q^u \,.\, c(q) > c(y) \Rightarrow u(q) > u(y)\}$.

2.2 Formal Mappings of Network Quality of Service

The mechanisms realizing 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. To rigorously relate the different viewpoints on network QoS, a well-defined translation of the specification is needed, called *QoS mapping*. The QoS mapping can be divided into

the QoS domain and the QoS scalability mapping.

The 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 the auxiliary functions $dm_P : Q_h \rightarrow P_l$ (performance mapping), $dm_R : Q_h \rightarrow R_l$ (reliability mapping) and $dm_G : Q_h \rightarrow G_l$ (guarantee mapping). A detailed description of the three mapping subfunctions is given in [1]. In general, the QoS mappings are neither injective nor surjective. That 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.

The QoS scalability mapping is needed to apply control aspects characterizing the dynamic adaptation of QoS parameters on different system levels. A QoS scalability mapping sm is a set of four mapping functions sm_{Util}, sm_{Cost}, sm_{Up} and sm_{Down}, translating the different scalability domains into each other. Due to limitations of space, we also omit a detailed description of the four functions. For a closer look, we refer to [1].

2.3 Meta-Model

In [2] we have introduced a first metamodel for network Quality-of-Service. Based on this metamodel, we derived a new one, which is also based on the formalization of network QoS surveyed in Section 2. The metamodel as shown in Figure 1 introduces a new class *NetworkQoSDescription*, that encapsulates QoS requirements specifications and QoS mappings. Additionally, QoS domains and QoS subdomains are aggregated in this class. This facilitates reuse since one subdomain can be used in several QoS domains, which is modeled by references. Moreover, the QoS specification is subdivided into a class *QoSSpecification* and *QoSProfile* to enable various application scenarios, *i.e.* use cases where the QoS domains are indistinguishable but the minimum and optimum QoS values resp. the description of the scalability are different. The relation between QoS specification and QoS domain is explicitly modeled. To simplify the implemtation, the domains of performance, reliability and guarantee are subsumed under a superclass *Domain*. Further, the belonging of the *ConcreteValues* to the one of the three domains is modelled as aggregations between *QoSValue* and *ConcreteValue* instead of introducing further subclasses. So the condition that a QoS value of a QoS domain contains only one concrete value is not visible in this metamodel. The *QoSScalabiliy* is exactly modelled as described in the formalization. The scalability consists of a *UtilityFunctions* and a *CostFunction* and two thesholds *up* and *down*. The three mappings for performance, reliability and guarantee are summed up in a new class *MappingFunction*. For every subdomain of the target domain, the *DomainMapping* contains one mapping function. The scalability mapping must not be explicitly modeled, as it is identical for all QoS requirement specifications.

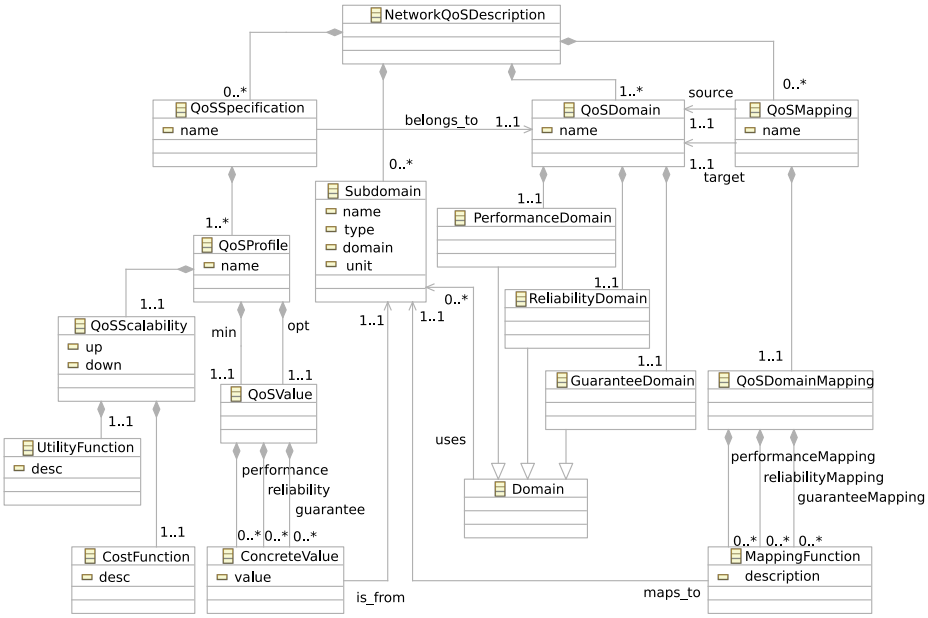

Fig. 1. Domain Model for the Graphical NQSL Editor

3 Network Quality-of-Service Specification Language

This section defines the formal *Network Quality-of-Service Specification Language (NQSL)*, a lexikal language for specifying the requirements on network Quality-of-Service. NQSL is a grammatical description of the formalization of network QoS and therefore powerful enough to host the concepts and criterias introduced in Section 2. It is a declarative language with a formal semantic since it is based on a mathematical model with functions (*e.g.* domain mapping dm) and operators (*e.g.* $\sim_u$).

In the following, we confine our presentation of the language definition to some excerpts. The complete definition of the language can be found in Appendix A. The syntactical notation of the presented language is given according to the Extended BNF (EBNF). The basics are briefly recalled to avoid ambiguities:

- non-terminals are written in angle-brackets `<non-terminal>`, terminals are enclosed by single quotes `'terminal'`
- productions are declared by `<non-terminal> = expansion;`
- sqare brackets enclose optional parts `[optional]`
- alternatives are separated by `|`

As described in Section 2.1, a QoS domain captures the QoS characteristics of a class of data flows, *i.e.* performance, reliability, and guarantee. A QoS domain

has a *name* to refer to and a *domain body* divided into declarations of performance, reliability and guarantee domains. These three declarations again are ordered sets of so called *subdomains*. The language definition for a QoS domain is:

```
<qosdomain_item>     = 'QoSDomain' <ident> <domain_body>;

<domain_body>        = '{' <partdomain_list> '}';

<partdomain_list>    = <partdomain_list> <partdomain_item>
                     | <partdomain_item>;

<partdomain_item>    = <pdomain_item>
                     | <rdomain_item>
                     | <gdomain_item>;

<pdomain_item>   = 'Performance' <partdomain_body>;
<rdomain_item>   = 'Reliability' <partdomain_body>;
<gdomain_item>   = 'Guarantee'   <partdomain_body>;
<partdomain_body> = '{' <subdomain_list> '}';

<subdomain_list>  = <subdomain_list> <subdomain_item>
                  |  <subdomain_item>;
```

As an example, the declaration of the QoS domain *Video* is given by

```
QoSDomain Video{

  Performance{
                (...)
  }
  Reliability{
                (...)
  }
  Guarantee{
                (...)
  }
}
```

The main building block of every QoS domain specification is the *QoS subdomain*. Subdomains are essential when defining multi-dimensional QoS domains. A QoS subdomain is identified by a unique *name* and specifies a *type* based on one or more basic data types (`integer`, `real`, or `enum`) or prior defined subdomains. Optionally, a definition of the *domain* of the type can be given to restrict to a set of possible values. The declaration of a QoS subdomain follows the next pattern:

```
<subdomain_item>  = 'SubDomain' <subdomain_body>
                  | 'SubDomain' <ident> ';';

<subdomain_body>  = '{' subdomain_desc:sdd '}'
                  | '{' /* EMPTY */ '}';

<subdomain_desc>  = <name_item> <type_desc>
                  | <type_desc> <name_item>;

<type_desc>       = <type_item>
                  | <type_item> <domainOfType_item>
                  | <domainOfType_item> <type_item>;

<name_item>          = 'name' ':' <ident> ';';
<type_item>          = 'type' ':' <datatype> ';';
<domainOfType_item> = 'domain ':' <domainOfType> ';';
```

As an example consider following QoS subdomain declaration:

```
Subdomain {
        name:   Resolution;
        type:   (Integer,Integer);
}
```

Here we have defined a subdomain of type *Resolution*. The possible values of this subdomain are pairs of *Integers*. By default, values of the *Integer* domain are not restricted, so every possible combination of integers is allowed.

To further restrict the possible values of a subdomain, a declaration of the domain can be given:

```
<domainOfType_item> = 'domain ':' <domainOfType> ';';

<domainOfType>  = <setofvalues>
                | <listofvalues>;

<setofvalues>   = '{' <intervalvalue_list> '}'
                | '{' <enum_list> '}';

<listofvalues>  = <intervalvalue_list>
                | <enum_list>;

<intervalvalue_list>  = <intervalvalue_list> ',' <intervalvalue_item>
                      | <intervalvalue_item>;

<intervalvalue_item>  = <interval>
                      | <value>;

<enum_list>    = <enum_list> ',' <ident>
               | <ident>;
```

A domain can be set of possible values or intervals. An interval is specified by a lower and upper boundary. The interval can include (`[..` resp. `..]`) or exclude the boundary (`]..` resp. `..[`). If no boundary is given, infinity is assumed.

An example is given below. To further restrict the possible values of the subdomain `Resolution` the domain is given allowing only three pairs of integer values:

```
subdomain {
        name:   Resolution;
        type:   (Integer,Integer);
        domain: {(320,240),(480,360),(640,480)};
}
```

If the *type* of a subdomain is `enum`, the declaration of the domain type is mandatory. In this case, the domain is a set of user-defined identifiers. Normally, the elements of an *enum* are ordered by a nominal scale (e.g. from 1 to 10). However, we assume that the first element declared has the lowest value.

```
<enum_list>    = <enum_list> ',' <ident>
               | <ident>;
```

For instance, an enumaration can be used to declare a subdomain describing the (available) degree of commitment:

```
Subdomain{
  name: DoC;
  type: enum;
  domain: {bestEffort, enhancedBestEffort, statistical, deterministic};
}
```

Hence, the performance domain of the QoS domain *Video* can be declared as:

```
  Performance{
      Subdomain {
        name:   Resolution;
        type:   (Integer,Integer);
        domain: {(320,240),(480,360),(640,480)};
      }

      Subdomain {
        name:   Quality;
        type:   Integer;
        domain: {25,50,75};
      }

      Subdomain {
        name:   FrameRate;
        type:   Integer;
        domain: [1,25];
      }
   }
```

To relate QoS requirements of different QoS domains, QoS domain mappings are needed. A QoS mapping initiated by the keyword *domainMapping* has a source domain *from* and a target domain *to*. Within a domain mapping, there are several mapping functions, translating higher layer QoS subdomains to lower layer subdomains or vice versa. If no mapping should be done, *i.e.* the subdomains are identical and the concrete values should stay the same, the mapping function can be left empty. In the following, the language definiton for QoS domain mappings is presented:

```
<mapping_part>  = 'domainMapping' 'from' <ident> 'to' <ident> <mapping_body>;

<mapping_body>  = '{' <mapping_list> '}';

<mapping_list>  = <mapping_list> <mapping_item>
                | <mapping_item> ;

<mapping_item>  = 'Performance' ':' <mappingfunc_list>
                | 'Reliability' ':' <mappingfunc_list>
                | 'Guarantee'   ':' <mappingfunc_list>;

<mappingfunc_list>  = <mappingfunc_list> <mappingfunc_part>
                    | <mappingfunc_part>;

<mappingfunc_part>  = <mappingfunc> ';';

<mappingfunc>       = <ident> '=' <expression>
                    | <ident>
                    | /*no mapping*/;
```

As an example consider the following mapping from the QoS domain *Video* to a QoS domain *Middleware*. The performance parameters are mapped to corresponding parameters *Frames* and *Period*, describing the number of middleware data frames of size *1420* Byte with an intervall *Period* seconds.

```
domainMapping from Video to Middleware{

  Performance:
    Frames=(160*Quality+3000)*(Resolution.1.-160)/(160*1420);
    Period=1/FrameRate;

  Reliability: (...)
  Guarantee:   (...)
}
```

The QoS requirements specification decribes the needs of a sevice user (application) on a communciation system. It is described by an unique *name* and is of a specific QoS domain, described by the keyword *uses*. The specification consists of several QoS requirement profiles, called *qosReq*. The QoS requirement profiles are described by *minimum* and *optimum* QoS values and a description of the *scalability*. The grammar of a QoS requirement specification is given below:

```
<specification_part>  = Specification <ident> 'uses'
                            <ident> <specification_body>;

<specification_body>  = '{' <qosreq_list> '}';

<qosreq_list>         = <qosreq_list> <qosreq_item>
                      | <qosreq_item>;

<qosreq_item>         = 'qosReq' <ident> <qosreq_body>;

<qosreq_body>         = '{' <qosreq_part> '}';

<qosreq_part>         = <minopt_list> <scalability_spec>
                      | <minopt_list>;

<minopt_list>         = 'minimum' <subspec_part> 'optimum' <subspec_part>
                      | 'optimum' <subspec_part> 'minimum' <subspec_part>;

<subspec_part>        = '{' <subspec_list> '}';

<subspec_list>        = <subspec_list> <subspec_item>
                      | <subspec_item> ;

<subspec_item>        = 'Performance' ':' <assignment_list>
                      | 'Reliability' ':' <assignment_list>
                      | 'Guarantee'   ':' <assignment_list>;

<scalability_spec>    = 'Scalability' <scalability_body>;
```

In the following example, the QoS specification *VideoTelephony* consists of one QoS profile *surveillance*, described by a minimum and optimum QoS value, as well as a description of the scalability.

```
Specification VideoTelephony uses Video {

    qosReq surveillance {

      minimum{       (...) }

      optimum{       (...) }

      Scalability{   (...) }

    }
}
```

The specification of a QoS value in detail is done by assigning concrete values to every subdomain of performance, reliability and guarantee. The scalability is described by an *utility* and *cost* functions and two thresholds *up* and *down*.

```
<subspec_part>        = '{' <subspec_list> '}';

<subspec_list>        = <subspec_list> <subspec_item>
                      | <subspec_item> ;

<subspec_item>        = 'Performance' ':' <assignment_list>
                      | 'Reliability' ':' <assignment_list>
                      | 'Guarantee'   ':' <assignment_list>;

<assignment_list>     = <assignment_list> <assignment_part>
                      | <assignment_part>;

<assignment_part>     = <assignment> ';';

<assignment>          = <ident> '=' <value>
                      | <ident> '=' <ident>;

<scalability_spec>    = 'Scalability' <scalability_body>;

<scalability_body>    = '{' <scalability_list> '}';

<scalability_list>    = <scalability_list> <scalability_part>
                      | <scalability_part>;

<scalability_part>    = <scalability_item> ';';

<scalability_item>    = 'util'   '=' <expression>
                      = 'cost'   '=' <expression>
                      | 'up'     '=' <simplevalue>
                      | 'down'   '=' <simplevalue>;
```

An example is given below. A requirements specification of the prior defined performance domain is given by assigning concrete values to the *Resolution* subdomain, the *Quality* subdomain, the *FrameRate* subdomain and the domain of the scalability. Therefore, the utility and cost functions have to be described by means of prior defined subdomains.

```
Performance:
          Resolution = (640,480);
          Quality = 75;
          FrameRate = 20;

(...)

Scalability {
        util = 0.3*(Resolution.1-160)/480 + 0.3* Quality/75 + 0.3*FrameRate/25;
        cost = Period*\#Frames*1420;
        up = 0.2;
        down = 0.1;
      }
```

4 Conclusion and Future Work

In this paper, we have presented *NQSL*, the *Network QoS Specification Language*, to formally specify network QoS. NQSL is derived from a previous formalization of network QoS requirements with specific emphasis of scalability and

cross-layer development. NSQL provides language elements for specifying QoS domains, QoS subdomains, and QoS mappings. Further, QoS requirements can be defined by specifying QoS profiles, expressed by minimum and optimum QoS values and a scalability value that consists of utility function, cost function, and two thresholds. The work presented in this paper solves a number of problems of practical relevance. First, it is very important that network QoS requirements be specified formally. While there are several languages reported in the literature already, NQSL goes one step further by supporting network QoS specification on all system layers, by including QoS scalability, and by supporting QoS mappings.

Our future work aims at developing a tool chain to support the efficient handling of NQSL specifications. These tools should include a graphical editor, an analyzer and an NQSL-to-SDL-generator. Second, an extensions of our tool chain for QoS system development, and SDL system designs that satisfy formally specified network QoS requirements are also planed.

References

1. Webel, C., Gotzhein, R.: Formalization of Network Quality-of-Service Requirements. In: Formal Techniques for Networked and Distributed Systems - FORTE 2007. Lecture Notes in Computer Science (LNCS) 4574, Springer (2007) 309–324
2. Schneider, D., Anastasopoulos, M., Bayer, J., Becker, M., Webel, C.: QoS Specification in Ambient Intelligence Systems. In: Proceedings of ICPS06/SEPS Workshop (SEPS'06), Lyon, France (2006)

A The Grammar

```
/* The grammar */

<qos_desc>  = <declaration_part>
            | <specification_part>
            | <mapping_part>
            | <declaration_part> specification_part>
            | <declaration_part> mapping_part> <specification_part>
            | <declaration_part> specification_part> mapping_part>;

/* Deklaration Part */

<declaration_part>  = <qosdomain_list>
                    | <subdomain_list>
                    | <subdomain_list> <qosdomain_list>;

<qosdomain_list>    = <qosdomain_list> <qosdomain_item>
                    | <qosdomain_item>;

<qosdomain_item>    = 'QoSDomain' <ident> <domain_body>;

<domain_body>       = '{' <partdomain_list> '}'
                    | '{' /* EMPTY */ '}';

<partdomain_list>   = <partdomain_list> <partdomain_item>
                    | <partdomain_item>;
```

```
<partdomain_item>    = <pdomain_item>
                     | <rdomain_item>
                     | <gdomain_item>;

<pdomain_item>  = 'Performance' <partdomain_body>;
<rdomain_item>  = 'Reliability' <partdomain_body>;
<gdomain_item>  = 'Guarantee'   <partdomain_body>;

/* Subdomain*/
<partdomain_body> = '{' <subdomain_list> '}'
                  | '{' /*EMPTY*/ '}';

<subdomain_list>  = <subdomain_list> <subdomain_item>
                  |  <subdomain_item>;

<subdomain_item>  = 'SubDomain' <subdomain_body>
                  | 'SubDomain' <ident> ';';

<subdomain_body>  = '{' subdomain_desc:sdd '}'
                  | '{' /* EMPTY */ '}';

<subdomain_desc>  = <name_item> <type_desc>
                  | <type_desc> <name_item>;

<type_desc>       = <type_item>
                  | <type_item> <domainOfType_item>
                  | <domainOfType_item> <type_item>;

<name_item>       = 'name' ':' <ident> ';';
<type_item>       = 'type' ':' <datatype> ';';
<domainOfType_item> = 'domain ':' <domainOfType> ';';

/* Types */

<datatype>        = <simpledatatype>
                  | <complexdatatype>;

<simpledatatype>  = <ident>
                  | 'Integer'
                  | 'Real';

<complexdatatype> = '(' <datatype_list> ')'
                  | 'Enum';


<datatype_list>   = <datatype_list> ',' <simpledatatype>
                  | <simpledatatype>;


/* Type Domains */

<domainOfType>  = <setofvalues>
                | <listofvalues>;

<setofvalues>   = '{' <intervalvalue_list> '}'
                | '{' <enum_list> '}';

<listofvalues>  = <intervalvalue_list>
                | <enum_list>;

<intervalvalue_list>  = <intervalvalue_list> ',' <intervalvalue_item>
                      | <intervalvalue_item>;

<intervalvalue_item>  = <interval>
                      | <value>;

<enum_list>   = <enum_list> ',' <ident>
              | <ident>;
```

```
/* Interval and Value */

<interval>  = <closed_interval>
            | <open_interval>
            | <left_open_interval>
            | <right_open_interval>;

<closed_interval>   = '[' <fromTo_item> ']'; // [1,2]
<open_interval>     = ']' <fromTo_item> '['; // ]1,2[
left_open_interval  = ']' <fromTo_item> ']'; // ]1,2]
right_open_interval = '[' <fromTo_item> '['; // [1,2[

<fromTo_item> = <value> ',' <value>  /* Value to Value Wert */
              |    ',' <value>       /* -infinity to Value */
              |        <value> ','   /* Value to +infinity */
              |    ','               /* -inf. to +inf. */

<value>       = <simplevalue>
              | <complexvalue>;

<simplevalue> = '+' <number>
              | '-' <number>
              |     <number>;

<number>      =  <Integer_Literal>
              |  <Real_Literal>;

complexvalue  = '(' <value_list> ')';

<value_list>  = <value_list> ','<value>
              | <value>;


/*Mappings*/

<mapping_part>  = 'domainMapping' 'from' <ident> 'to' <ident> <mapping_body>;

<mapping_body>  = '{' <mapping_list> '}';

<mapping_list>  = <mapping_list> <mapping_item>
                | <mapping_item> ;

<mapping_item>  = 'Performance' ':' <mappingfunc_list>
                | 'Reliability' ':' <mappingfunc_list>
                | 'Guarantee'   ':' <mappingfunc_list>;

<mappingfunc_list>  = <mappingfunc_list> <mappingfunc_part>
                    | <mappingfunc_part>;

<mappingfunc_part>  = <mappingfunc> ';';

<mappingfunc>       = <ident> '=' <expression>
                    | <ident>
                    | /*no mapping*/;

/* Specification */

<specification_part>  = Specification <ident> 'uses'
                            <ident> <specification_body>;

<specification_body>  = '{' <qosreq_list> '}';

<qosreq_list>         = <qosreq_list> <qosreq_item>
                      | <qosreq_item>;

<qosreq_item>         = 'qosReq' <ident> <qosreq_body>;
```

```
<qosreq_body>          = '{' <qosreq_part> '}';

<qosreq_part>          = <minopt_list> <scalability_spec>
                       | <minopt_list>;

<minopt_list>          = 'minimum' <subspec_part> 'optimum' <subspec_part>
                       | 'optimum' <subspec_part> 'minimum' <subspec_part>;

<subspec_part>         = '{' <subspec_list> '}';

<subspec_list>         = <subspec_list> <subspec_item>
                       | <subspec_item> ;

<subspec_item>         = 'Performance' ':' <assignment_list>
                       | 'Reliability' ':' <assignment_list>
                       | 'Guarantee'   ':' <assignment_list>;

<assignment_list>      = <assignment_list> <assignment_part>
                       | <assignment_part>;

<assignment_part>      = <assignment> ';';

<assignment>           = <ident> '=' <value>
                       | <ident> '=' <ident>;


<scalability_spec>     = 'Scalability' <scalability_body>;

<scalability_body>     = '{' <scalability_list> '}';

<scalability_list>     = <scalability_list> <scalability_part>
                       | <scalability_part>;

<scalability_part>     = <scalability_item> ';';

<scalability_item>     = 'util'  '=' <expression>
                       = 'cost'  '=' <expression>
                       | 'up'    '=' <simplevalue>
                       | 'down'  '=' <simplevalue>;

<expression>           = <expression> '+' <expression>
                       | <expression> '-' <expression>
                       | <expression> '*' <expression>
                       | <expression> '/' <expression>
                       | <expression> '\%' <expression>
                       | '-' <expression>
                         \%prec UMINUS
                       | '(' <expression> ')'
                       | <number>
                       | <ident>;

/* Literals and Identifier*/

<Integer_Literal> = [0-9]+;

<Real_Literal>     = 0'.'0
                   | [0-9]*'.'[0-9]+
                   | [0-9]*('.'[0-9]+)?[eE][+-]?[0-9]+;

<ident>            =  [a-zA-Z._] [0-9a-zA-Z._-]*;
```

B Example: VideoTelephony

```
subdomain {
  name: ResponseTime;
  type: Int;
  domain: [0,...];
  //unit: sec.;
}

subdomain {
  name: AudioSamplingRate;
  type: Int;
  domain: {11025, 22050, 44100,48000};
  //unit: Hz;
}

subdomain {
  name: AudioResolution;
  type: Int;
  domain: {8, 16, 24,32};
  //unit: Bit;
}

subdomain {
  name: VideoResolution;
  type: (Int, Int);
  domain: {(320, 240), (480,360),(640, 480)};
  //unit: (px, px);
}

subdomain {
  name: VideoFPS;
  type: Int;
  domain: [1,...];
  //unit:  ;
}

subdomain {
  name: PictureQuality;
  type: Int;
  domain: [0,100];
  //unit: %;
}

subdomain {
  name: DoC;
  type: Enum;
  domain: {bestEffort, enhancedBestEffort, statistical, deterministic};
  //unit:  ;
}

subdomain {
  name: DataLoss;
  type: Int;
  domain: [0,100];
  //unit: % max.;
}

subdomain {
  name: Priority;
  type: Int;
  domain: [0,];
  //unit:  ;
}

subdomain {
```

```
  name: Synchronisation;
  type: Enum;
  domain: {lipSynchronous};
  //unit:  ;
}

subdomain {
  name: Performance;
  type: Int;
  domain: [0,...];
  //unit: max. pro min;
}

subdomain {
  name: Throughput;
  type: Int;
  domain: [0,];
  //unit: kb/s;
}

subdomain {
  name: Delay;
  type: float;
  domain: [0,];
  //unit: max. sec;
}

subdomain {
  name: Jitter;
  type: float;
  domain: [0,];
  //unit: max. sec;
}

subdomain {
  name: PacketLoss;
  type: Int;
  domain: [0,100];
  //unit: %;
}

subdomain {
  name: Corruption;
  type: Int;
  domain: [0,100];
  //unit: %;
}

subdomain {
  name: OrderPreserving;
  type: bool;
  domain: {yes,no};
  //unit:  ;
}

subdomain {
  name: Stat;
  type: Int;
  domain: [0,100];
  //unit: %;
}

QoSDomain SessionCall {

    performance {

        subdomain ResponseTime;
```

```
            subdomain Performance;
        }
        reliability {
            subdomain DataLoss;
        }
        guarantee {
            subdomain DoC;
            subdomain Priority;
            subdomain Stat;
        }
    }
    QoSDomain Audio {
        performance {
            subdomain AudioSamplingRate;
            subdomain AudioResolution;
        }
        reliability {
            subdomain DataLoss;
        }
        guarantee {
            subdomain DoC;
            subdomain Priority;
            subdomain Stat;
        }
    }
    QoSDomain Video {
        performance {
            subdomain AudioSamplingRate;
            subdomain AudioResolution;
            subdomain VideoResolution;
            subdomain VideoFPS;
            subdomain PictureQuality;
        }
        reliability {
```

```
        subdomain DataLoss;
        subdomain Synchronisation;
    }
    guarantee {
        subdomain DoC;
        subdomain Priority;
        subdomain Stat;
    }
}
QoSDomain System {
    performance {
        subdomain Throughput;
        subdomain Delay;
        subdomain Jitter;
    }
    reliability {
        subdomain PacketLoss;
        subdomain Corruption;
        subdomain OrderPreserving;
    }
    guarantee {
        subdomain DoC;
        subdomain Priority;
        subdomain Stat;
    }
}
domainmapping from Video to System {
  performance:
    Throughput =  (VideoResolution[0]*VideoResolution[1]*0,62*VideoFPS
                    *(PictureQuality/2+0,5)+AudioSamplingRate
                        *AudioResolution/8)*1,5/1024;
    Delay = 2;
    Jitter = 0,5;

  reliability:
    PacketLoss = DataLoss;
    Corruption = DataLoss/3;
    OrderPreserving = yes;

  guarantee:
    DoC = Guarantee;
    Priority = Priority;
```

```
    Stat = Stat;

}

specification VideoTransmission uses Video {

    qosreq VideoTransmission {
      minimum {
        performance:
          VideoResolution = 320,240;
          PictureQuality = 50;
          VideoFPS = 5;
          AudioResolution = 8;
          AudioSamplingRate = 11025;

        reliability:
          Synchronisation = lipSynchronous;
          DataLoss = 10;

        guarantee:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = low;

      }
      optimum {
        performance:
          VideoResolution = 640,480;
          PictureQuality = 90;
          VideoFPS = 25;
          AudioResolution = 16;
          AudioSamplingRate = 44100;

        reliability:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = low;

        guarantee:
          Synchronisation = lipSaynchronous;
          DataLoss = 2;

      }
      scalability {
        util: (...);
        cost: (...);
        up: 0,2;
        down: 0,2;
      }
    }

}

specification AudioTransmission uses Audio {

    qosreq AudioTransmission {
      minimum {
        performance:
          AudioSamplingRate = 11025;
          AudioResolution = 8;

        reliability:
          DataLoss = 5;

        guarantee:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = medium;
```

```
      }
      optimum {
        performance:
          AudioSamplingRate = 44100;
          AudioResolution = 16;

        reliability:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = medium;

        guarantee:
          DataLoss = 5;

      }
      scalability {
        util: (...);
        cost: (...);
        up: 0,2;
        down: 0,2;
      }
    }

}

specification SessionCall uses SessionCall {

    qosreq SessionCall {
      minimum {
        performance:
          ResponseTime = 5;
          Performance = 3;

        reliability:
          DataLoss = 0;

        guarantee:
          Priority = medium;
          DoC = enhancedBestEffort;
          Stat = 90;

      }
      optimum {
        performance:
          ResponseTime = 5;
          Performance = 3;

        reliability:
          Priority = medium;
          DoC = enhancedBestEffort;
          Stat = 90;

        guarantee:
          DataLoss = 0;

      }
      scalability {
        util: (ResponseTime>5?0:1-ResponseTime/6)*(DataLoss>0?0:1);
        cost: ;
        up: 1;
        down: 1;
      }
    }

}

specification EmergencyCall uses SessionCall {
```

```
    qosreq EmergencyCall {
      minimum {
        performance:
          ResponseTime = 1;
          Performance = 3;

        reliability:
          DataLoss = 0;

        guarantee:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = high;

      }
      optimum {
        performance:
          ResponseTime = 1;
          Performance = 3;

        reliability:
          DoC = enhancedBestEffort;
          Stat = 90;
          Priority = high;

        guarantee:
          DataLoss = 0;

      }
      scalability {
        util: (ResponseTime > 3?0:1-ResponseTime/3,3)*(DataLoss>0?0:1);
        cost: ;
        up: 1;
        down: 1;
      }
    }
}
```