13

Towards Automatic Trading of QoS Parameters in Multimedia Distributed Applications[1]

*P. Dini, A. Hafid**
Computer Science Research Institute of Montreal
1801, McGill College Street, #800, Montreal, (Qc), H3A 2N4, Canada
e-mail: dini@crim.ca
**Université de Montréal, Département d'Informatique et de Recherche Opérationnelle, Montréal, H3C 3J7, Canada*
e-mail: hafid@iro.umontreal.ca

Abstract

Several existing and emerging studies exhibit demands for the *definition, measuring, testing,* and *enhancement of QoS*. Since *real-time multimedia applications* require *guaranteed performance services* or *graceful degradation services*, many problems are particularly complex. ODP considers QoS aspects embedded within the trading function. Emerging systems portray particular service parameters, which must be considered by an ODP trader. The advent of multimedia leads to *new functional services* (guaranteed, predictive), *new management services* (QoS management, synchronization management), *new protocols* (negotiation, re-negotiation), and *new mechanisms* (parameter translation, parameter matching). This paper deals with the matching function and particular trading aspects in multimedia distributed applications. In order to offer a more flexible matching and reduce the negotiation dialog, we propose a formal model for service parameters, defining special parameter fields. Each new feature is analysed across multimedia examples, its particularities, its value space, and the customer-provider issues related to it. Based on this model, a precise definition of the matching function which is useful to automate the negotiation of QoS is proposed. An algorithm implementing this function is analysed. A model for the evaluation of contractual QoS at the parameter and service level, and specific formulas are presented.

Keywords

Multimedia distributed applications, QoS trading, QoS matching, automatic negotiation

1 INTRODUCTION

Managing QoS is a complex task because dynamic changes or unexpected behaviors due to users and to resource disponibilities are often possible within open distributed systems. User requests, as well as interactions of system components need to be defined, controlled, and monitored. Commonly, the notion of quality of services (QoS) is used to appreciate the interaction between two system entities. QoS is used in ODP-based systems to perform the service trading.

Several existing and emerging studies exhibit demands for the *definition, measuring, testing,* and *enhancement of QoS*. Current solutions partially cover some areas, such as *open distributed systems*,

1. This work was supported by a grant from the Canadian Institute for Telecommunication Research (CITR), under the Networks of Centres of Excellence Program of the Canadian Government

J. Rolia et al. (eds.), *Open Distributed Processing and Distributed Platforms*

multimedia applications, and *distributed systems (DS) management. Real-time multimedia applications* require *guaranteed performance services* or *graceful degradation services*. The advent of multimedia leads to *new functional services* (guaranteed, predictive), *new management services* (QoS management, synchronization management), *new protocols* (negotiation, re-negotiation), and *new mechanisms* (parameter translation, parameter matching). The support of a distributed multimedia application involves several system components which provide services under several constraints. These constraints are usually expressed as parameter values associated with the provider services (provider performance), and by parameter values specified by customers (customer desired QoS).

In order to automatically manage multimedia applications within large distributed systems such as Common Work Supported Computing or Teleconferencing, several relevant ways are concerned with precise definition of QoS, translation relations and matching functions, and mechanisms to update, compute and control QoS parameters. In this paper we highlight the matching and QoS issues related to the function of matching.

The current packet-switching networks providing the best-effort service are no longer adequate for multimedia distributed applications having stringent performance requirements in terms of *throughput, delay jitter, delay* and *loss rate*. Consequently, the functional and management perspectives must be re-considered. For the former, two types of services have been proposed to cope with particular requests of multimedia applications namely, the *guaranteed service* model and the *predicted service* model (Zhang and Ferrari 1994). Additionally, the advent of multimedia applications generates new management requirements with respect to the QoS, such as QoS definition, QoS model, QoS evaluation, and QoS management (Bochmann and Hafid 1997). The major problem is how to evaluate the QoS in order to appropriately manage multimedia applications and which rules govern such a manipulation. In multimedia applications the service parameters could be offered at *different satisfactory levels* with respect to *requested constraints*. Hence, *the services translation, parameter translation,* and *matching the parameter values* are important mechanisms. *Translation* signifies agreements between correspondent names of non-identical parameters or names of non-identical services, whereas *matching* refers to agreements between values of identical or similar parameters. The result of these two operations defines the QoS of a *customer-provider cooperation*.

Although the following basic terms are generally applicable for any kind of interactions between system components, we use multimedia examples through our paper, since multimedia distributed applications are more sensitive to the QoS aspects. QoS refers notably to *service parameters*, e.g., delay jitter, throughput, and audio-quality. Each multimedia component has its own services which display *property values,* as prescribed by its specification. However, running systems offer only current values of these parameters. Commonly, these values continuously vary, periodically, or non-periodically, affecting the performance of the provider. This aspect represents a service *degradation* and it is commonly known as *QoS degradation*. Multimedia applications consist of one or many *customer-provider relations* which cooperate to a common target, as requested by the end-user. In a *customer-provider relation*, QoS is defined by service parameters of the provider (called the *system performance -SP-* at the *QoS provider interface*) which satisfy customer requests (*QoS customer interface*).

Each time a relation is established, parameter values of the requested services are identified and compared with performance parameters of the provider. The process is called *negotiation* and its result is a *cooperation contract*. In order to initialize a contract, the partners could *re-negotiate* customer demands with respect to the provider current performance. Negotiation and re-negotiation use the operation called *matching the parameter values. Matching* the parameter values is a cornerstone operation because of the diversity of parameters and their value spaces (Dini and Bochmann 1994). When this operation is performed, the *contractual values* between a customer and its provider become a basis to further evaluate the QoS. Our present paper focuses on the matching aspects.

The paper presents an well-defined approach to perform the *matching of parameter values*. Section 2 presents some achievements and new challenges of existing solutions, and highlights different development directions. In Section 3, a precise model for a service parameter and a matching function are presented. An algorithm for applying the matching function is described. A model for the evaluation of QoS at the parameter and service level is presented in Section 4. We conclude with respect to our proposal, the state of the implementation, future related work, and an appendix concerning a concrete example of using the model presented in this paper.

2 MOTIVATION

Although the QoS is a global feature of component interactions, it becomes mostly relevant in critical systems. Among these systems, multimedia distributed applications raise several constraints concerning the *synchronization*, *delay*, or *jitter*, and other particular service constraints referring to the *audio quality*, *video color*, or *image clarity*, which are not common to usual applications. On the other hand, multimedia applications portray two features. i.e. (i) the service parameters of these applications are well-defined for each kind of media, and (ii) the value space for each parameter is finite and well-known. When multimedia features are enhanced gradually, the value space may vary, but its range is always known.

2.1 Related work

The *matching function* is generally used in many domains, mainly in linguistics (speech recognition), chemistry (molecular structures), and computers (traces, parameters, types). Speech recognition is phonemes-based, molecular structures are fundamental structures-based, whereas traces are event or state sequence-based. The operation of matching a certain sample into a large space could have invariants which are non-negotiable. The present study refers to the matching of types and parameter values. In the classical programming, the types of the current and formal parameters of a function must match. For example, if the function is declared as f (x: REAL, y:INTEGER), and a: INTEGER and b: INTEGER, the invocation f (a, b) is wrong. Nowadays, the arrival of object-oriented programming allows to use the parameter subtyping. If the types of parameters are valid in the conformance tree, the function could be invoked (polymorphism). The implementation code of such a function is the same for INTEGERS and REALS. Consequently, with respect to types, a matching function has a certain maneuverability within the subtyping tree, but the choice is not negotiable.

Moreover, an instance of a type is externally visible across its interface signature (operations). The matching function is used to semantically identify the type the instance belongs to. Wang and Archer (1994) have proposed five stages of matching which are *driven by similarities*, using the syntax and semantics definition of an object. As a result, matching is performed between the requested object and the classes implementing the model (ibidem). Since usually the system model is well-defined, their approach is static. There is no negotiation, but a certain flexibility degree of matching is given by semantics similarities. Matching mechanisms are equally used in conformance test specification and QoS testing methodologies. For example, the conformance test specification language TTCN (Tabular and Tree Combined Notation) uses constraint matching mechanisms: a constraint ASP (Abstract Service Primitive) parameter or PDU (Protocol Data Unit) field shall match the corresponding received ASP parameter or PDU field if the received parameter or PDU field has exactly the same value to which the expression in the constraint evaluates (Sarikaya and Wiles 1991). Constraints are predicate conditions concerning values or ranges. We will take the range approach to perform a flexibility of matching in multimedia applications. Montiel and Najm (1994) have built a TTCN-based language to test the QoS for multimedia services, where constraints on QoS parameters could be explicitly declared. Because up to date the matching has been performed strictly from functional perspectives, it is difficult to enhance QoS from the ODP Management perspective in the same framework. The current approaches do not favor the negotiation and re-negotiation. We propose a model of service parameters concerning the QoS which is more flexible. This model could be also fruitful to evaluate transitions in a QoS degradation model (Hafid and Dini 1995), QoS itself (Section 4), or the quality of system configurations (Dini and Bochmann 1995).

2.2 New challenges in multimedia applications

We distinguish two kinds of services in multimedia applications, i.e. *interactive services,* such as *news-on-demand*, and *cooperating services* between two multimedia components of a system, called hereafter *customer-provider cooperation relations*.

In a news-on-demand multimedia example presented by Kerherve, Vogel, Bochmann, Dssouli, Gecsei, and Hafid (1994), the user intention is modelled by several parameters offered as choices, e.g. *<media-type, text-format, audio-quality, color, size, frame-rate, delay, cost>*. Each tuple parameter has a value space from which the user may select *one value at a time.* For example, the value space of the color could be *<black&white, gray, unclearcolor, clear color, super-color>*. A concrete user choice could be, in this case, the tuple *<video, gray, small, reduced-rate, 2 sec, 5$>*. In news-on-demand services, each matching failure is followed by a negotiation or re-negotiation of parameter values in order to accommodate user needs to the provider performance. In this case, the provider must display its current parameter value space (called current-space). The cooperation is performed only with the user agreement via several re-negotiations. Re-negotiations are performed also when user needs or the provider performance change.

In customer-provider relations, the quality requested by the customer and the performance offered by the provider must be appropriately matched. In automatic trading systems, there are no human decisions as a catalyst; consequently, particular details on each service parameter must be specified to allow automation. These details could refer to *type of parameter*, e.g., negotiable or non-negotiable, *additional fields*, e.g., the measure unit, and *values to be negotiated*, e.g., requested value with possible relaxations, or performance value with additional constraints.

We claim that it is not sufficient to only know the value space and the measure unit for a service parameter to automatically monitor the QoS in trading-based multimedia systems. In the sequel, we present a parameter model which can be used to develop server applications which automatically adapt QoS according to user needs.

3 MATCHING QoS PARAMETERS

This section presents an informal and formal model for service parameters and a formal definition of the *matching function* in the context of QoS. In our approach, customer-provider cooperations are driven by three functions. The *trading function*, called **R** function, identifies a possible provider for a well-requested multimedia service by a customer needing precise QoS. The *translation function*, called **T** function, signifies agreements between correspondent names of non-identical parameters or name of non-identical services, whereas the *matching function*, called **M** function, refers to agreements between values of identical or similar parameters.

3.1 Component model

Negotiation and re-negotiation of customer-provider agreements lead establish cooperations or to changes original contractual agreements. This implicitly signifies variations of QoS which represent a violation of contractual QoS, that is, either a degradation, or an enhancement. We assume that a system component could always be represented by its embedded features (*ef*), its exported features (*ex*) consisting of offered services possibly with some constraints belonging to them according to its own performance, and imported features (*in*) depending on the performance of the possible provider, that is $C = <in, ef, ex>$. We are concerned with *in* and *ex* features. Usually *in* and *ex* features are services (*s*) described by performance parameters (*p*). For example, the service set offered by a component C can be specified as $ex_C = <s_1, s_2, ..s_m>$, where each service has a parameter set $s_i = <p_{i1}, p_{i2},..p_{ik}>$. Usually, for a media provider component, $m=1$, and k is finite. For example, parameter set of an *audio* media-type provider is defined by the following basic tuple *<audio-quality, delay, cost>*, whereas the user, viewed as a media customer, could be described by a single *in* service called `request`, whose parameters are *<media-type, text-format, audio-quality, color, size, frame-rate, delay, cost>*.

Each *in* or *ex* parameter has a value space which must be guaranteed (or updated) by the provider part, and considered by the customer part. For example, the *audio-quality* could be within the space *{phone, cd, tape, radio}*, the cost could be within [a$, b$], and the *delay* could vary into [a sec, b sec]. Also, the media-type could have the following *value space {text, image, audio, video, audio&video, composed&types}*. Embedded features *ef* refer to service availability, component reliability, and the

internal behavior of a component.

3.2 Precise definition of a service parameter

The matching function assumes that there is no conflict (due to the translation function) between service names. We have distinguished three classes of fields used to match customer-performer parameters for a well-known service namely, *parameter identification fields* (parameter name, value space, measure unit, parameter type), *matching condition fields* (variation index, necessity index), and *effective matching fields* (value type, parameter value, possible variations). Consequently, in our matching model, the service parameter is a 9-tuple p = <n, t, v, vt, cs, u, pv, vi, nx>, where

```
n  = parameter name;          u  = measure unit;
t  = parameter type;          pv = possible variations;
v  = parameter value;         vi = variation index;
vt = value type;              nx = necessity index, and
cs = current value space;
```

as summarized in the following. A complete description and examples are given in Dini, Hafid and Bochmann (1995).

• `parameter name (n)`: The parameter name identifies a service parameter. A video offers a single service called *video service* which is characterized by the parameter set *{color, frame-rate, size, delay, cost}*. Since each media equipment has a finite set of services, one can easily define the value space of the parameter name. The matching function assumes that the parameter names at the customer and provider are identical. If two parameters have the same semantics or are equivalent (consequently, they could be matched) but they have distinct names, the translation function ensures this information to the matching function. The parameter name is of type STRING.

• `parameter type (t)`: The parameter type defines if the parameter value may be or not negotiable. If a *screen-type* parameter of the *device-service* offered by an operating system (see Appendix) has the value of 8-bit-gray, there is no possibility to relax it because a device has a well and unique technology, while if the *supported-format* parameter has the current value, let's say *ascii*, a negotiation may be possible, e.g., *postscript*. If a service evolves, the parameter type changes. For example, an old device-service can offer the *color = {black&white}*, thus, the color parameter is not negotiable at the level of service. New devices may have *color = {black&white, color, super-color}* and consequently, the parameter *color* is negotiable. Other parameters are negotiable by their nature, e.g., *cost, delay*. Obviously, a non-negotiable parameter does not support re-negotiations. In the case of contractual failures because of a non-negotiable parameter, the function R must be activated at the management level (for trading other components with appropriate requested parameters). The parameter type of provider services are more relevant with respect to the contract agreement at the negotiation level. At the management level, this feature is relevant for the customer across the trading function R. In our model, $t \in$ *{negotiable, non-negotiable}*.

• `parameter value (v)`: The parameter value specifies the current requested value of the parameter. A requested value could be either a single value, an interval, or a set. A precise value could be *cost = {8}[$]*, an *well defined interval*, e.g. the *throughput* parameter of *software services* is within *[α, β][frames/sec]*, or a *finite set*, e.g. the parameter called *format* at the *multimedia customer interface* could be within *{ascii, postscript, gif, JPEG, MPEG, DVI}*. This value may be negotiated and re-negotiated. For a customer, this value represents the requested (desired) value, whereas for a provider this parameter has no significance. The parameter value of a customer must fit on the current-space value of the provider equivalent parameter.

• `value type (vt)`: Value type represents the basic type of a value among the following types: INTEGER, REAL, STRING, INTERVAL, SET. SET could contain either, INTEGER, REAL, or STRING values. The *throughput* or *size* parameters of the *storage-service* at a database-interface

must be INTEGER [bytes/sec] or INTEGER [bytes] respectively, the *jitter* and *delay* are REAL [sec], whereas the *guarantee-class* is STRING. This is an important additional information that allows to detect any non-matching. The reason is that when a negotiation is performed on parameters having different types, either a fault has occurred somewhere, or the customer does not know exactly the provider services (trading errors). The information is used at the management level by the R function. If a relation of *type conformance* is defined by a subtyping hierarchy between value types, the M function must use this relation instead of the identity of value types. If a customer presents $vt = \{\emptyset\}$, any provider's type is accepted. Contrary, if the provider specifies $vt = \{\emptyset\}$ the customer can request a value of any type. In our model, $vt \in$ *{INTEGER, REAL, STRING, INTERVAL, SET}*.

- `current value space (cs)`: The *current value space* represents the space where the parameter value can vary, that is, a unique value, a set, or an interval. The value space established by the specification of a system is called *basic-space*. However, the performance of a component is rarely as prescribed by the specification. Consequently, each media device updates (*current-space*) the value space of each of service parameters, following its current performance. The matching is performed on this *current space*. For example, the *screen* parameter's value space is *{1-bit, 8-bit,-gray, 8-bit-color, 24-bit-color}*. For the reason of quality, a customer can investigate and select the best provider, let us say, *screen = {24-bit-color}*. But, if the potential output device list is reduced to *{1-bit, 8-bit-color}*, the negotiation and contractual agreements are based on this *current-space*. The *current-space* is important in large distributed systems, where the degradation and enhancement of the *current-space* frequently occur. This space is currently updated according to the enhancement of provider services. The current-space is relevant for the provider part, since rarely *current-space= basic-space* (commonly, *current-space* is included in *basic-space*). The customer must accommodate his/her requests to the current-space constraints of the provider. This feature is relevant at the negotiation and management level. At the negotiation level, it defines the boundary of the negotiation, whereas at the management level it allows to select between similar providers offering a larger current value space. Generally, *current values spaces* $\subseteq$ *basic-space*. Parameters may have *precise values*, e.g. the *delay* parameter at the server-interface, or internal values, e.g. the *cost* at the transport service interface. In this case, we consider that the current value space is of the form [min, max] of type INTERVAL. If the current value space is not continuous, the SET type may be considered.

- `measure unit (u)`: Matching parameter values requires identical measure units. Consequently, the matching function must convert measure units. The measure unit defines the quantification of the parameter value. This quantification could be either a *standardized unit* (second, $), a *non-standard-quantitative unit* (frame/sec, bytes/sec), or a *linguistic unit* (acceptable, good). There are parameters which commonly have the *unique unit* at any interface, e.g. *delay* (sec) or *cost* ($), whereas others do not, e.g., the *throughput* parameter is measured in frames/sec at the operating system interface, in bytes/sec at the server interface, and in TSDU/sec at the transport-service-interface. Other parameters have not a unit of measure, e.g. the *audio-quality* has its value space {phone, cd, tape, radio}. Linguistic units, such as *acceptable, good, well,* and *best* could be a subjective measure only at the level of human customer. Finally, some parameters have a value space, but they have not an measure unit, e.g., a database interface offers the services *find-records* having the guarantee *class = {guaranteed}*. The measure unit is important for both customer and provider parts. There is no value space for this feature. However, if the measure unit is standardized, the customer and provider must use appropriate divisions or multiples of it (e.g., sec, ms, Mbytes, Kbytes, Mbit), or composed standardized units (e.g., frames/sec, Mbytes/sec). For the linguistic units, the value space must not only have the same semantics, but also be identical.

- `possible variations (pv)`: The possible variations represent values which can modify the parameter value of a customer, or the current value space of a provider. They determine a stronger or looser negotiation with respect to the value matching. A customer may accept exactly a *cost* of β $ prescribed in its parameter value field of the cost parameter. The *cost* could be [0 $, $\beta + \beta_1$ $] or [0$, $\beta - \beta_2$ $], i.e. the customer accepts a possible variation of [0 $, β_1 $], or [-$\beta_2$ $, 0 $], or it

requires a *super-color* value for the color parameter, but it could accept *normal-color*. If a provider has an instable performance, it could specify the *current-value-space* for a parameter by highlighting observable or non-observable variations. For example, if the frame-rate at the user interface has the value *frame-rate* = {TV-rate = 25 images/sec}, a possible non-observable variation is frame-rate = {TV-rate}, but with the possible variation field pv = <1, 1>. That is, the negotiated *frame-rate* could be *frame-rate = [24, 26] images/sec (one says pv- =1, pv+=1)*. As the parameter value could vary by increasing or decreasing its *v* with the possible variations, the syntax of these modifications are <pv+, pv-> with the following semantics. For the customer part, <pv+, pv-> represents a possible variation of requirements, as presented by the parameter value. For the provider part, <pv+, pv-> defines current-space values which can be offered. If the requirements become permanent, the current value space is appropriately updated, and the field of possible variations becomes <∅, ∅ >.

At the customer side, pv+ represents a relaxation for the negotiation, while pv- defines supplementary constraints which must be first considered by the management system, at the level of customer *parameter value*. If the management system does nor find a provider offering *v* composed with *pv-*, then it searches for a provider offering *v*, if not, *v* composed with *pv+*. At the provider side, pv+ represents a possible enhancement of the provider performance, while pv- represents a possible degradation of its performance at the level of *current-value-space*. Contractual agreements negotiated across these fields may success or not. Thus, the management system must imperatively monitor these cooperations. If a provider improves its performance, its current-value-space is updated, while pv becomes <∅, ∅>.

The following relations are considered: (i) for the providers, possible variations *(provider)* ⊆ *basic-value-space (provider), and (ii)* for the customers, *possible variations (customer)* ⊆ *current-value-space (provider)*. In the case of providers, these possible variations are initially specified. In the case of customers, these variations depend on initiator constraints, that is, if one accepts or not some variations, and on the current-value-space of the invoked provider. Because of the previous inclusions, pv+ and pv- could have the type INTEGER, REAL, SET, or INTERVAL.

- `variation index (vi)`: The variation index is a BOOLEAN parameter feature which validates or invalidates the use in the negotiation mechanisms of the values contained in the feature *possible variations*. We revisit one of the previous example. If the *frame-rate* at the user interface has the value *frame-rate* = {25 images/sec}, a possible non-observable variation is frame-rate = {TV-rate}, but with the possible variation field r = <1, 1>. That is, the negotiated *frame-rate* is *frame-rate = [24, 26] images/sec*. However, if the image is used by a device whose sensitivity exceeds that of a human, non-periodical images (sometimes cloudy) are captured. Consequently, a customer needs sometimes temporary to invalidate the *pv* field. Even further, the management system must evaluate the QoS at different interfaces. Since it is possible that the use of the *pv* field damages other cooperations, this *vi* could be driven also by special management monitors. *false/true* decision is primarily customer or provider dependent. It could be also monitored by a special guard. For example, a customer could accept an over *cost* during the work-week on some services, but not in the week-end, while for some providers the situation may be the opposite. In our model, *ni* ∈ *{true, false}*. *true* indicates that the customer/provider can use the value recorded in its possible variation field.

- `necessity index (nx)`: The necessity index is of BOOLEAN type which describes whether the parameter is absolutely requested or not. It refers rather to the necessity of this parameter as a field of a requested service. For example, if a customer prescribes a *delay* and a *cost* with *nx = {false}*, the managers will perform the connection even they have not find an appropriate provider (offering the *cost* ={}). The customer has no idea of the *delay* to prescribe a possible variation, but the managers could keep this information in order to eventually find an appropriate provider in the future. After a number of cooperation tentatives, the customer can evaluate an approximative parameter value, and it could change the necessity index or update the *possible variations*. For such a negotiation, a contractual agreement is first rapidly achieved, then, the negotiation managers evaluate the possibility of the enhancement for this achieved QoS. Parameters which are not satisfied in the negotiation, but have *nx = false* are not considered in the evaluation of the QoS. This feature is considered

only for the customer side. Usually, this item refers to non-functional or not absolutely requested parameters. In opposition with the possible variation field, their refusal does not affect the QoS of the customer-provider cooperation. In our model, $nx \in \{true, false\}$. *False* indicates that the customer can ignore this parameter.

3.3 Formal definition of the matching function

We say that two components *a* and *b* offer services each other if there are relations R, T, and M such as $p_{ik}Mp_{jl}$, s_iTs_j, and aRb imply that $\exists s_i \in ex_a$ *and* $\exists s_j \in in_b$ such that, for i and j, and for $\forall k$ and $\forall l$, the relations R, T, and M hold. We present in Figure 1 the matching aspects related to our parameter model into a customer-provider relation. We have indicated the matching fields and the fields which are arguments of the M, T, and R previously introduced (the *nx* field could *eventually* be used by the T and R). We construct now the *matching function* considering the definitions of parameter fields. The relation between *pv* and *vi* fields is highlighted because of their closed relation. They could be considered by the R function in order to select the best potential providers.

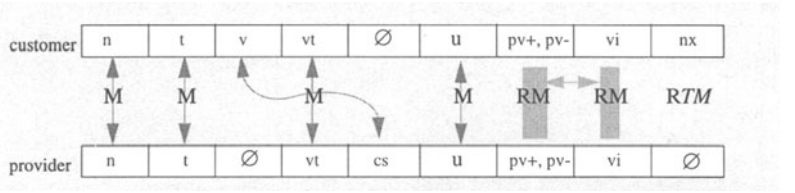


Figure 1 Correspondence of parameter model fields in a customer-provider relation

Let us suppose that each service parameter is described by the previous fields, and there are R and T such that aRb and s_iTs_j. Let us take the appropriate parameters called p_{ik} and p_{jl} and analyze their matching within our model. Without the loss of generality we consider that *i* and *j* refer respectively to the customer, and provider. The following matching rules must govern the matching in our model.

`Rule1:` If s_iTs_j and $p_{ik}Mp_{jl}$, then $s_i.p_{ik}.n$ =T=> $s_j.p_{jl}.n$

This proposition assumes that the names of two parameters are the same or similar. This information is transmitted to the M function by the T function.

`Rule2.` If $s_i.p_{ik}.u \neq s_j.p_{jl}.u$, then either $\exists u1$ such as $s_i.p_{ik}.u1 = s_j.p_{jl}.u$, or $\exists u2$ such as $s_i.p_{ik}.u = s_j.p_{jl}.u2$, or $\exists u1$ and $\exists u2$ such as $s_i.p_{ik}.u1 = s_j.p_{jl}.u2$.

In the case when the two parameters have not the same measure unit, this proposition assumes that the T function provides equivalent measure units. Since the possible *name similarity* takes place at the T level, this operation can not be performed at the M level. In the case of name identity, the M function can perform the equivalence of measure units. The M function keeps the transformation tables for different measure units.

`Rule3:` If a subtyping hierarchy is not defined, the M must verify whether $s_i.p_{ik}.vt = s_j.p_{jl}.vt$. Contrary, the M function must apply the *type conformance* to validate the request. Declarations of type $vt = \{\emptyset\}$ on the customer or provider side allow to ignore type conformance or type identity constraints.

`Rule4:` The negotiated value requested as a quality (**Q**) by a customer is either *v*, or *v* composed with *pv*, according to *vi*, let's say $\mathbf{v_{neg}}$.

The composition follows normal rules of the "$\cap$", "\", and "$\cup$" operations defined on sets and intervals (a unique value is considered as a set of one element). The last operation corresponds to *pv+*, whereas the remainder to *pv-*.

`Rule5:` The value-space offered by a provider as its performance (**P**) is either *cs*, or *cs* composed with *pv*, according to *vi*, let's say $\mathbf{cs_{neg}}$.

The composition follows the "$\cap$", "\", and "$\cup$" operations defined on sets and intervals (a unique value is considered as a set of one element). The last operation corresponds to *pv+*, whereas the

remainder to *pv*-.

`Rule6` : The final matching is obtained if $p_{ik}.\mathbf{v_{neg}} \subseteq p_{jl}.\mathbf{cs_{neg}}$.

The combinations shown in Figure 2 are possible, i.e. the requested quality **Q** and the offered performance **P** could have a unique value, be a set, or be an interval.

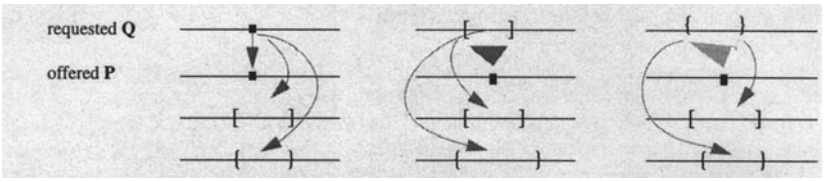


Figure 2 Matching possibilities

In the first case, to get a parameter matching, either the requested value is identical with the offered value, e.g. critical multimedia cooperation, or the requested value belongs to the value interval or value set performance. Otherwise, the parameters do not match.

In the second case, we have a matching if, either the offered value is within the requested interval, or the intersection of requested interval with offered interval or offered set is not empty.

Finally, if the offered value is within the requested set, or the intersection of requested set with offered interval or set is not empty, the matching is achieved.

`Rule7` : If the *nx* = *{false}* the result of the matching could be ignored, but the result must be communicated to the requester and the QoS managers.

As a result, the requester may adapt his/her requested Q, while the managers may optimize the trading function R by recording the best performers.

`Rule8` : If the parameter type verifies *t* = *{non-negotiable}* for either the customer or provider part, the manager must be informed in order to improve the R function.

3.4 General algorithm for matching QoS parameters

A general algorithm corresponding to the specific parameter fields and considering the previous rules is presented in Figure 3. The algorithm assumes that the name constraint of Rule1 holds. The necessity index *nx* is tested and the matching is going on. If the *nx* = *{false}* the manager is informed, according to Rule7. The first step of verifications concerns the parameter type *t*, as prescribed by Rule8. Further, the unit transformations are performed and the type conformance or type identity, if there is no subtyping hierarchy, is validated. Let us suppose that the problem of unit transformations is performed according to Rule2. If the typing conformance is a success (Y), the algorithm continues; otherwise, the manager receives a N information and the algorithm is stopped. However, both the customer and provider may inhibit the type conformance block, as presented by Rule3.

According to Rule4, the subsequent action computes the largest negotiable requested value, and further, the current-value-space is updated as prescribed by Rule5. The operator notation ■ holds for the composition between current value *v* and possible variation *pv*, and respectively between current-space *cs* and its possible variations *pv*. The last operation is to verify the inclusion of requested values versus offered current-value-space, as presented in Rule6. Different combinations are presented to the manager, in order to evaluate the QoS established. In the next section we present a proposal for the quantification of the QoS based on this algorithm.

4 EVALUATING CONTRACTUAL QoS

As a response to a customer service request, the invoked provider may satisfy the request constraints, or not. In the last case, the customer is acknowledged, whereas the trader selects across the R function another potential provider. Otherwise, a contractual achieved QoS is performed, that is, the constraints

of all requested service parameters are satisfied. We evaluate the degree of this satisfaction, first, for a parameter, and then, for the service.

4.1 Contractual QoS at the parameter level

Each customer invocation of a provider's service implies the use of the matching function for every service parameter specified by a customer. For the moment, we focus only on the parameter. The value requested as a quality (**Q**) by a customer is compared to the current-value-space offered by a provider as its performance (**P**). We note with P^+, P^-, Q^+, Q^-, $Q^{-/+}$, and $P^{-/+}$ the cases where the matching has been achieved by composing offered *current-value-space* or *requested value* with their possible variations.

`Proposition 1:` P^+, P^-, and $P^{+/-}$ correspond to performance enhancements, whereas Q^-, Q^+, and $Q^{-/+}$ represent possible relaxations of the requested value, that is, **Q** constraints are less strong.

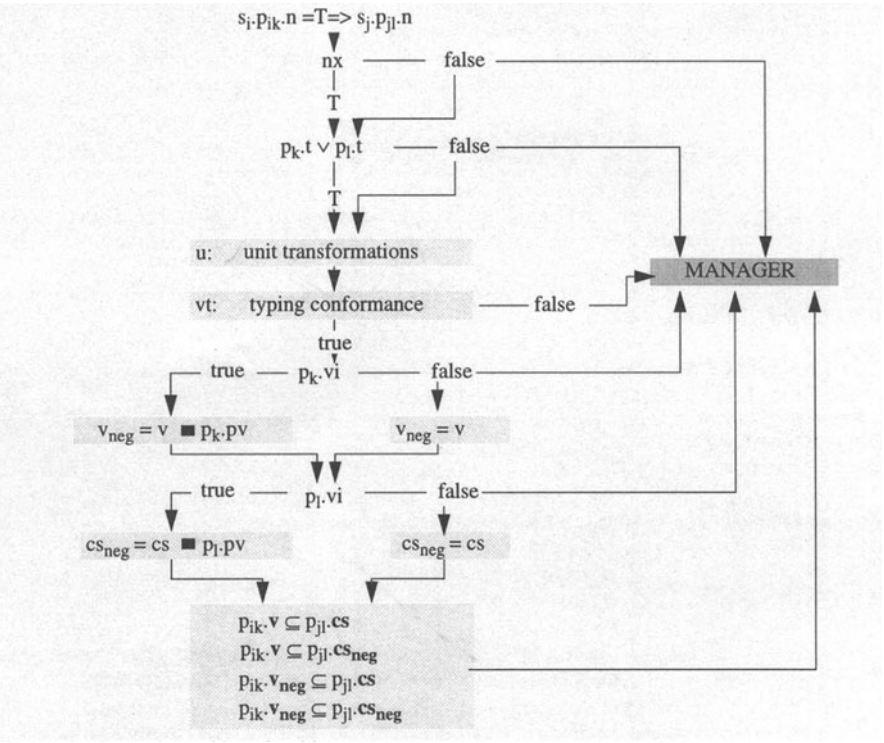


Figure 3 General matching algorithm

`Proposition 2:` If the requested quality **Q** is achieved across the performance **P**, we say that **P** satisfies **Q**, and plot this relation as P ➤ Q. Consequently, the relation ➤ is defined on the Cartesian product A = {P, P^+, P^-, $P^{+/-}$} X {Q, Q^+, Q^-, $Q^{+/-}$}.

`Proposition 3:` The QoS is reversely proportional with the number of possible relaxations via *pv* fields, that is, the partners' qualitative efforts are proportional with the number of +/- signs involved in the ➤ relation. Considering the relaxation costs equal for customer and provider, and independent for *pv+* or *pv-* variations, we identify five classes of initial contractual QoS, as presented in Figure 4. The class number 5 is the best. The class 5 defines a *rightMatching* type of matching,

whereas the class 1 defines a *minusMatching*.

`Proposition 4` : We introduce the function E: (➤) --> {1, 2, 3, 4, 5}, defined in Figure 4, as the evaluation of the satisfaction relation ➤. Consequently, the QoS at the parameter level has the value

$$QoS_{parameter} = E(➤) \quad (I)$$

Q \ P	P	P^+/P^-	$P^{+/-}$
Q	0/5	1/4	2/3
Q^+/Q^-	1/4	2/3	3/2
$Q^{+/-}$	2/3	3/2	4/1

LEGEND:

a/b: a = number of relaxations within the ➤ relation;
b = the evaluation of the achieved contractual QoS within the range 1÷5.

Figure 4 A possible evaluation of achieved contractual QoS

Consequently, in our model, each requested service parameter of a customer-provider cooperation contract has a first evaluation of its achieved QoS. This allows us to globally evaluate the contract at the service level, and second, to measure the QoS degradation, or QoS enhancement.

4.2 Contractual QoS at the service level

Our model for the evaluation of contractual achieved QoS for a given service is strictly based on the number of service parameters involved in the matching function having the necessity index nx = *{true}*. Let us suppose now that we have a service s_i with k parameters respecting the previous condition.

`Proposition5` : If a service has m_i parameters with $QoS_{parameter} = i$, we define the evaluation of the achieved contractual QoS at the service level as

$$QoS = (\sum m_i \times i) / k, \text{ where } k = \sum m_i \quad (II)$$

The range of QoS defined by this formula is [1, 5]. The maximum value is obtained when all parameters have the QoS class of 5, whereas the minimum when all parameters have the class 1.

`Proposition6` : Composing the formulae I and II we can get the weight w of one parameter p_{ik} on the QoS of the service s_i to which it belongs to.

$$w(p_{ik}/s_i) = QoS_{parameter} / QoS \quad (III)$$

This weight is significant for the QoS managers in order to especially monitor and survey those parameters which can dramatically affect the QoS, i.e. their weight is predominant.

5 CONCLUSION

In this paper we have presented an approach to model service parameters in order to facilitate the matching between requested constraints of a customer and the performance offered by a provider. We have highlighted several QoS aspects particularly significant for the multimedia applications and proposed a formal and informal definition of parameter fields. For each field, we have argued by multimedia examples the field utility, its particularities with respect to the negotiation protocols, its distinct meaning for customer and provider, and its value space. Based on this parameter description we have build a set of rules for the matching function, supposing that the translation function has furnished a prior analysis. We have presented an algorithm implementing the matching function conforming to our model. Further, based on our parameter model, we have evaluated the contractual QoS at the parameter and service levels. The evaluation criteria are related to possible variations of the requested quality or offered performance, according to specific parameters fields which validate or invalidate these extensions. Five QoS classes at the parameter level have been identified, with the hypothesis that relaxation costs are equal for both customer and provider, and independent of the sense of variations. Finally, considering only non-optional parameters, we have proposed a QoS evaluation at the

service level.

The matching model is going to be melt into a QoS degradation model of news-on-demand services in multimedia applications (Project CITR) (Hafid and Dini 1995), and applied to evaluate the quality of the automatic (re)configuration in management policies across distributed systems (Project IGLOO) (Dini and Bochmann 1995). The experimental platform in the CITR project focuses on multimedia aspects and consists of a network employing point-to-point links coupled to a high speed ATM switch to form an ATM LAN and two power PCs IBM RS/6000 running AIX that support compatible ATM host interfaces (Bochmann, Kerherve, Hafid, Dini and Pons 1996). The implementation is performed in C++. The experimental platform of the IGLOO project refers to the automatic reconfiguration across distributed systems. The structure of the platform is: (1) three interconnected LANs using DEC 3000/4000 stations, (2) the OSIMIS version 1.0 which uses ISODE to operate an upper management stack, and (3) an object-oriented management database build following our architecture.

Current extensions refer to QoS classes based on distinct costs at the provider or customer side (Dini, Hafid and Bochmann 1996), and also on different weights of the new offered current-value-space or requested quality evolution. The matching function and its implementing algorithm will be equally adapted.

6 REFERENCES

Bochmann, v.G., Hafid, A. (1997) Some Principles for Quality of Service Management, Accepted for publication in *Distributed System Engineering Journal*, 1997

Bochmann, v. G., Kerherve, B., Hafid, A., Dini, P., Pons, A. (1996) Architectural Design of Adaptive Multimedia Systems, *IEEE Workshop on Multimedia Software Development,* Berlin, Germany, March 25-26 1996.

Clarck, D., Shenker, S., Zhang, I. (1992) Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism, *Proceedings of ACM SIGCOMM'92*, Baltimore, Maryland, August 1992.

Dini, P., Bochmann, v. G., (1994) *Multi-level QoS negotiation in Multimedia Distributed Systems*, IGLOO Technical Report, CRIM, Montreal, 1994.

Dini, P., Bochmann, v. G. (1995) Design of a Configuration Management Subsystem based on OSI/ ODP Approaches, Incorporating Self-Control Mechanisms to Guarantee the Survivability of the Networks, IGLOO Technical Report, CRIM, Montreal, 1995.

Dini, P. Hafid, A., Bochmann, v. G. (1995) Matching QoS Parameters in Multimedia Applications, *Technical Report CITR/ University of Montreal*, Montreal, 1995.

Dini P., Hafid, A., Bochmann, v. G. (1996) Modeling QoS Multimedia Costs in Distributed Systems, *The 1996 Pacific Workshop on Distributed Multimedia Systems*, Hong Kong University of Science and Technology (HKUST), Hong Kong, June 25-28, 1996, pp. 238-245.

Ferrari, D., Verma, D. (1990) A Scheme for Real-Time Channel Establishment in Wide-Area Networks, *IEEE Journal on Selected Areas in Communications,* 8(3), pp.368-379, April 1990

Hafid, A., Dini, P. (1995) QoS Degradation Model, *CITR Technical Report*, University of Montreal, Montreal, Canada.

Kerherve, B., Vogel, A., Bochmann, v.G., Dssouli, R., Gecsei, J., Hafid, A. (1994) On Distributed Multimedia Presentational Applications: Functional and Computational Architecture and QoS negotiation, *Proceedings of High Speed Networks Conference*, Vancouver, Canada, 1994, pp. 1-19

Montiel, J., Najm, E. (1994) A QoS Testing Methodology for Multimedia Services, *Proceedings of Workshop on Distributed Multimedia Applications and Quality of Service Verification*, Montreal, Canada, May-June 1994.

Sarikaya, B., Wiles, A. (1991) Standard Conformance Test Specification Language TTCN, *Technical Report, University of Montreal,* 1991.

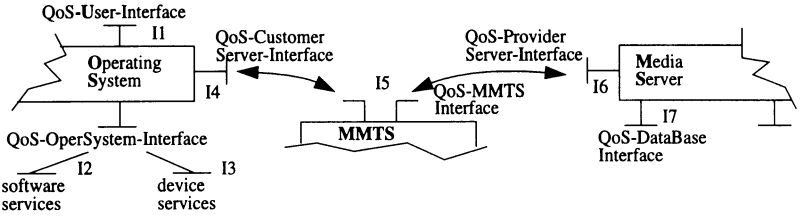
Zhang, H., Ferrari, D. (1994) Improving Utilization for Deterministic Service in Multimedia Communication, *Proceedings of the International Conference on Multimedia Computing and Systems,* Boston, Massachusetts, May 1994.

Wang, S., Archer, N. P. (1994) Identifying Inheritance Structure in Object-Oriented Systems Analysis: A Pattern Matching Approach, *JOOP,* May 1994.

Appendix

Partial representation in our model of the example on *news-on-demand* presented in Kerherve, B., Vogel, A., Bochmann, v. B., Dssouli, R., Gecsei, J., Hafid, A. (1994).

General architecture:



Four partners: **U** (user), **OS** (operating system), **MS** (media server) **MMTS** (multimedia transport server

1. **U** (**in**, don't care, don't care)
   in ∈ {audio-request, video-request, image-request, text-request, combined-of-them}
with the services
   audio-request = <audio-quality, delay, cost>;
   video-request = <color, frame-rate, delay, cost>;
   image-request = <size, color, delay, cost>;
   text-request + <text-format, delay, cost>,
where the parameters are
   cost: [0, MAX] [$];
   delay: [0, MAX] [second];
   text-format: {ascii, postscript, frameMaker, ..,..}[-];
   color: {black&white, gray, color, super-color}[-];
   size; {normal, 640xN, 480xN, MAX}[-];
   frame-rate: {frozen-image, reduced-rate, TV-rate}[-];
   audio-quality: {telephon, cd, radio, tape}[-]

2. OS (**in**, don't care, don't care)
   in ∈ {device-services, software-services, I4-services}
with the services
   device-services = <class-guarantee, audio-device, disk-drive, supported-type>
where the parameters are
   class-guarantee: {guaranteed, best-efort, predictive, reservation}[-];
   audio-device: {telephone, cd, radio, cd, tape}[-];
   disk-drive: {-}[bytes];
   support-type: {text, image, audio, video, combinations}[-];
and
   software-devices = <class-guarantee, supported-format, throughput, delay>

where the parameters are
class-guarantee: {guaranteed, best-effort}[-];
support-format: {ascii, postscript, gif, tiff, MPEG, DVI,..}[-];
throughput: [0, MAX][frame/sec]
delay: [0, MAX] [second].

3. **MS** (don't care, don't care, **ex**)
ex ∈ {audio-offer, video-offer, image-offer, text-offer, combined-of-them}
with the services
video-offer = <format, size, throughput, packet-size, jitter, delay, cost, guaranteed-class>
where the parameters are
format: {JPEG, MPEG, DVI}[-];
size: [0, MAX][bytes];
throughput; [0, MAX][Mbit/sec];
packet-size: [0, MAX][bytes];
jitter: [0, MAX][sec];
delay: [0, MAX][sec];
cost; [0, MAX][$];
guaranteed-class: {guaranteed, best-effort}[-]

4. MMTS (don't care, don't care, **ex**)
ex ∈ {transport-multi-service}
with the services
transport-multi-service = <TSDU$^{\alpha}$–max-size, throughput, delay, jitter, cost, guarantee-class, reliability>,
where the parameters are
TSDU-max-size: [0, MAX][byte];
throughput: [0, MAX][TSDU/sec];
reliability: {reliable, error-rate}[-];
guarantee-calss: {best-effort, guaranteed}[-];
jitter: [0, MAX][sec];
delay: [0, MAX][sec];
cost: [0, MAX][-].