Representation of Quality of Service Preferences by Contract Hierarchies

by

Christian Becker*, Kurt Geihs*, and Jan Gramberg*

1. Introduction

In an open electronic service market, competing service providers offer services to clients who may choose freely among the service types and service providers. Before the service provision, a client needs to identify a particular server and negotiate a service contract that determines the contractual constraints for the service interactions. Part of the contract will be an agreement on the service quality that is expected by the client and that will be provided by the server. For example, a stock exchange information service may have QoS parameters such as actuality of information, service availability, guaranteed response time, and cost. Obviously, QoS parameters are application specific and their actual values depend on the current state of the whole distributed system, e.g. on the load condition of the server, on the response time of the sub-services that are employed by the information service, on the network thruput, and so on. QoS inherently is an end-to-end issue, i.e. the client's perception of QoS depends on the QoS of many activities that occur in the context of the service interaction and that together are responsible to guarantee e.g. a certain response time level. In our project MAQS (Management Architecture for Quality of Service) we develop and implement a comprehensive framework for QoS management in distributed object systems. The framework will support all three main phases of QoS management: negotiation, monitoring/adaptation, and accounting/profiling.

In this paper we concentrate on the QoS negotiation that is needed before the service interactions can start. In Section 2 we will first describe our assumptions and the resulting system model. Section 3 discusses requirements and alternatives for the specification of client preferences. In Section 4 our solution is presented that is based

* Dipl.-Inform. Christian Becker, research assistant at the distributed system unit, dept. of computer science, University of Frankfurt/Main, Germany

Prof. Dr. Kurt Geihs, head of the distributed system unit, dept. of computer science, University of Frankfurt/Main, Germany

Jan Gramberg, graduate student at the distributed system unit, dept. of computer science, University of Frankfurt/Main, Germany

on hierarchical refinement of contracts. Finally, we briefly explain how the negotiation will be integrated into our overall QoS framework, and how our work will proceed.

2. System Model

A Service offers a specific functionality to clients. Clients access these functions by calling methods on the service's functional interface. The non-functional issues of the service interaction, i.e. the QoS handling, is accessible to clients via a conceptually separate QoS interface.

The service interaction is divided into three phases:

1. Session Establishment: Client and service negotiate about the QoS characteristic which shall be delivered and the concrete values of the QoS.

2. Interaction: Monitoring ensures that the delivered QoS is conformable to the negotiated QoS. Renegotiation or adaptation provide mechanisms to adapt to better or worse QoS.

3. Accounting: Service interaction, especially if it is QoS enabled, is related to resource consumption. Charging clients based on the delivered QoS seems more fair than flat rates. Additionally this prevents a monopolization of resources, since higher resource consumption is related to higher costs. Thus, clients are likely to reduce their resource consumption to the actual needed resources.

The QoS of a particular service depends on the current status of the overall distributed system. In general there will be a choice of different QoS levels that can be provided to a client in a certain situation. Therefore, if a client is willing to accept more than one QoS level, the service can choose which one will be offered to the client. Since individual clients may have different QoS preferences, a negotiation of QoS has to take place when a client binds to a server. This negotiation must respect the client preferences.

3. Specification of Client Preferences

This section introduces client preferences into the negotiations between clients and servers. Different kinds of representations are discussed. As a result of the discussion

we introduce a simple yet powerful approach for the specification and implementation of client preferences.

3.1. Quality of Service

QoS represents the non-functional aspects of service interaction betweena client and a service. In contrast to the functional view on a service, which is the same for all clients, the non-functional, QoS-view depends on the individual expectations of a client. The following terminology, which was introduced by the ISO[1] and adopted by the OMG[2], will be used throughout the rest of this paper. A QoS characteristic isolates a distinct non-functional aspect of interaction. QoS characteristics are abstract entities in the sense that they are quantifiable from a client's perspective but are no concrete entities in the system, e.g. latencies cannot be determined exactly but be approximated by measurements. A QoS characteristic is represented in the system by a set of quantifiable QoS parameters. An actual state of a QoS characteristic in the system is called a QoS level. A collection of QoS characteristics addressing the same domain, e.g. real-time, fault-tolerance, and security, is called a QoS category.

3.2. QoS Agreements

A negotiated QoS level between a client and a service is called a QoS agreement. A client may work properly with a variety of QoS levels. But among these levels some QoS states may be preferred to others. During negotiation, a client could either request QoS levels one by one consecutively until a suitable level has been acknowledged by the server, or offer his complete list of preferences to the service which chooses the one that best matches the current constellation. The first case implies heavy network activity especially when the number of possible QoS levels is high, while the second case demands an aggregated representation of client preferences.

3.3. Representing Client Preferences

A number of specification techniques for representing client preferences in QoS negotiations are conceivable. Some of them have been suggested already in the literature.

[1] ISO (1995)

[2] OMG (1997)

3.3.1. Enumeration

A simple straight forward representation of client preferences could be the enumeration of all possible parameter combinations in the preferred order. Clearly, this list would explode in its size with only few parameters. A parameter representing bandwidth in a numerical variable would produce a list with all possible values of this variable. This could be reduced by choosing „better“ representations like intervals. However, most QoS management architectures do not support the definition of user defined QoS parameters. Therefore, unsuitable QoS parameters can not be changed. Thus a separate representation of client preferences seems necessary either when they are not integrated in the system up front or the QoS parameters are inadequate for the representation of client preferences.

3.3.2. Preference Relations

Utility theory provides a notation of preference relations. A preference relation is a directed graph of nodes which model a state and indicate via edges the states which are preferred to this one. As with enumeration and ordering of states the number of states might explode. On the other hand the resulting datastructure is intuitively clear and easy to handle for human beings and machines as well. A separate notation for preferences which maps to QoS parameters might help reducing the number of states and therefore increase the usability of this approach.

3.3.3. Worth Functions

Worth functions provide a simple yet powerful solution to the problem of state explosion of the previous mentioned approaches. A worth function operates on the representation of a QoS characteristic, i.e. the QoS parameters, and delivers as a result value the client's preference of this QoS instance. This helps to reduce the size of the transmitted data, since there are only representations of functions to be transmitted for each QoS characteristic, a client is interested in. However, providing expressiveness to clients leads to complex mathematical expressions as candidates for worth functions. Constructing these complex functions is a non trivial task, which would have to be done by users or application object developers. Hence, worth functions lack ease of applicability though they reduce the number of states and thus allow compact expression of client preferences.

3.3.4. Mapping to Vectorspaces

Reducing the complexity of worth functions by restricting them to linear functions leads to a mapping of QoS parameters into a vector space. Each dimension, e.g.

component of a QoS parameter, is mapped to a linear function. The notion of regions as introduced in QuO[3] corresponds to a vectorspace. Another approach[4] allows user defined metrics in a vector space for user defined ordering of states. Thus, the state of a QoS parameter is related with a point in a vector space. Distances between points can be determined and an implicit ordering exists. This is easy to use and compacts the representation of client preferences as well. This approach is applicable when QoS dimensions and possible client preferences upon these are of linear nature. In generic QoS management, such assumptions about QoS dimensions can not be made. Therefore this approach lacks universality.

3.3.5. Preference Language

One approach to reduce the number of possible states in client preferences could be a mapping from a specification of client preferences to a QoS parameter. Mapping from a higher abstraction to the QoS parameter can hide system dependent QoS parameters, e.g. a quality of {good, medium, low} could be mapped to intervals of a QoS parameter for bandwidth. The user is not confronted with system or implementation specific QoS parameter representations. Thus, the preference specification is decoupled from a specific implementation of a QoS characteristic. Designed up front with structuring mechanisms for preference specification this could provide a uniform approach for existing systems, since the QoS representation in the system is mapped from the preference language to the underlying QoS parameter.

3.3.6. Summary

Nowadays QoS management architectures do not provide notions for client preferences. Service offers are based on implicitly coded assumptions. This approach is viable for fixed sets of a priori known QoS parameters. In generic QoS management architectures this can not be done since clients may not share the same preference when a set of offers is presented. Thus an additional notion for client preferences has to be provided. Using QoS parameters to specify client preferences as well is not feasible since the number of resulting states will explode. Thus other representations have to be provided. Depending on the underlying QoS architecture, these can be kept simple. However, aiming at generic QoS management requires a uniform representation of client preferences independent of the underlying QoS parameters. A preference language can provide the necessary abstractions.

---

[3] Zinky, J.A. et.al. (1997)

[4] Linnhoff-Popien, C./Thießen,D. (1997)

3.4. An Example

A service *S* offers financial data, like stock market data, to clients. A QoS characteristic may be the *actuality* of the data which reflects the aging of the transmitted financial information. Some applications, e.g. purchase decisions, may need the most actual data whereas statistic computations can rely on less actual data. Better actuality, i.e. high quality, will be more expensive than less quality. The QoS parameter for such a service is a structure containing two parameters for actuality and the corresponding cost. Depending on their application requirements clients may have different profiles which QoS they prefer. Hence a service must know the preferences of a client to offer the best matching QoS according to the clients perspective. If there are many (actuality, price) levels offered, the enumeration of client preferences will fail.

In the following section we will present our approach of defining client preferences. By using hierarchical grouping of states which are determined by so called rejectment-levels, an refinement based mechanism is introduced to specify the order of preferences. The refinement in conjunction with so called rejectment levels helps decreasing the number of states in a client preference. Our approach is as powerful as worth functions but far easier to use.

4. Hierarchical Preference Relations

Client preferences are expressed by hierarchically grouping states which determine a *rejectment*, i.e. a client accepts all states of a distinct QoS characteristic till they reach a certain bound. Such an approach satisfies the following objectives:

- generic: i.e. it can be applied to arbitrarly defined QoS parameters;
- multi-category: i.e. not tied to the requirements of just one or a few specific QoS categories;
- easy to use: i.e. allows an intuitive ordering of desired states.

Clients may use the specification to indicate preferences to a service, while services may use it to specify their offers to clients.

4.1. Rejectment Levels

A broad variety of QoS characteristics from several QoS categories has been analyzed by Koistinen[5]. It is remarkable that these QoS characteristics can be modelled with only a few constructs. They introduce the notion of a *dimension* which allows the specification of a part of a given QoS characteristic. Dimensions are typed (numerical, set, enumeration) and are ordered. The order of a dimension can be user defined. A QoS parameter is the aggregation of all dimensions necessary to specify the state of the QoS in the system. A QoS characteristic is represented by its QoS parameter. Additionally, each QoS characteristic provides a *base contract*. The base contract is a collection of abstractions (i.e. dimensions) operating on a QoS parameter. The base contract must be mapped to the corresponding QoS parameter, thus the negotiation can decide whether the given contract can be fulfilled. The client can refine this base contract to specify a less general state by assigning a certain value to a dimension and by ordering the dimension. For example, in some applications, a buffer might be considered in a preferable state, when being full (e.g. audio buffering) or in others when it is empty. Thus a dimension specifying the filling of a buffer must be ordered in the distinct meaning. Any refinement of a base contract defines a *rejectment level*. Informally this means, the client specifies the least acceptable state of the dimensions with respect to the order of the dimension.

```
base_contract actuality-qos{
  numerical price;
  enumerate update{actual, daily, weekly};
}
```

The above given base contract[6] for actuality specifies two dimensions. One is of numerical type for the price and corresponds to the price of delivering a certain actuality specified by the dimension update.

```
refined_contract n1 refines actuality-qos{
  price=10 with Order "<";
  update=daily with Order {actual < daily < weekly};
}
```

The refined contract *n1* constraints the base contract, thus only QoS with a lower price than 10 (in whatever currency like Euro, USD) and an actuality specified better than `daily` are allowed.

Refined contracts can be refined as well. As a result, a hierarchie of contracts is described. For a preference relation the states described must be ordered, thus an evaluation can determine which state reflect the client's most prefered one.

---

[5] Frølund, S./Koistinen, J. (1998)

[6] This is an example for refining and does not reflect the concrete syntax.

4.2. Hierarchy of Contracts

Refining contracts leads to a graph with a root - the base contract. Each node in a refinement hierarchy may only refine one contract. Thus the resulting structure is a tree. Figure 1 shows a tree as a result of multiple refinements of a base contract.

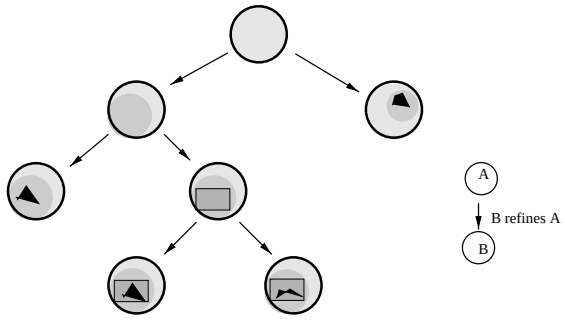

Fig. 1 refinement hierarchies

The resulting tree can be interpreted as a subtype relation based on refinement[7]. A mapping from contracts to QoS parameters has to be provided in order to evaluate the state of such a refinement in the system and the semantics of refinement have to be specified to achieve an ordering of contracts. Then the question is: when does a node in the tree define a state prefered to another node?

4.3. Partial Order of Client Preferences

In order to allow the specification of preferences a rule has to be applied which determines the preferred states. A node in the tree defines a refinement of the parent node. Thus, the child node is more restrictive with respect to the order of the dimensions. The associated values define a rejectment level. Each dimension of a node specifies a lower bound with respect to an order. Acceptable states for a node have smaller values in all dimensions. The most desired states are represented by the lowest leaf in the tree.

Therefore the rules of modelling client preferences are:

Rule of Preference

For each node reflecting a state ϕ which is prefered over states $\{\Psi_1, ..., \Psi_n\}$ it has to be ensured that $height(\phi) > height(\Psi_i)$ holds for $i \in \{1 ... n\}$.

Rule of Indifference

[7] Liskov, B./Wing, J. (1993)

For all nodes reflecting states $\{\rho_1, ..., \rho_n\}$ which are not prefered over each other it has to be ensured that $height(\rho_i) = height(\rho_j)$ with $i,j \in \{1, ..., n\}$.

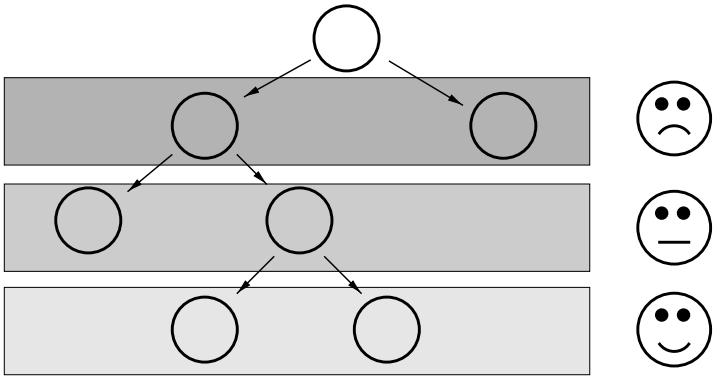

Fig. 2 indifference and preference

Figure 2 illustrates the levels of indifference in the tree. If two nodes ϕ_1, ϕ_2 have the same height in the tree, i.e. $height(\phi_1)=height(\phi_2)$, the client is indifferent about them. A service might choose which to deliver if both are possible.

Since it does not matter, which QoS of an indifference level is offered, the traversal of the tree can be done in several strategies. A service might implement a strategy to offer the least resource consuming QoS to satisfy a maximum of clients or use a simple first fit strategie on traversal and offers the first satisfied QoS on the path. Figure 3 illustrates possible paths through a tree.

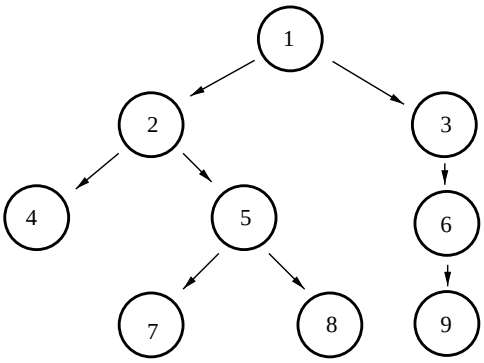

Fig. 3 tree traversal paths

Possible paths must only ensure that before the next level in the tree is inspected all nodes below this level are visited. Hence, three valid paths could be:

Path 1	7	8	9	6	5	4	2	3	1
Path 2	7	9	8	5	4	6	3	2	1

Path 3	8	7	9	4	5	6	3	2	1

During visiting the nodes, it has to be checked whether the specified rejectment is fulfilled by the system. Therefore a mapping from the node specification to the QoS parameters in the system has to be established. The order of nodes visited may vary as long as the rules of indifference and preference are obeyed.

4.4. Neu Classification of Contract Hierarchies

As stated before our approach provides the same expressiveness as worth functions. A worth function maps a value v_i to each state $\underline{S}$ of a QoS parameter *S*. Two cases have to be isolated. If the mapping is isomorph the range of values is disjunct. Thus the corresponding contract hierarchie will be a list with nodes reflecting the states in order of the related value. The leaf is associated with the contract describing the state with the highest (most desired) value. If the mapping is not isomorph, states $\underline{S}_1$ and $\underline{S}_2$ with the same value have to modelled thus they are in the same height in the tree to respect the Rule of Indifference. As a result the contract hierarchy will form a tree not a list.

However, providing the same expressiveness without additional benefits would not legitimate yet another preference representation. The users of QoS enabled communications are objects provided from object implementors. These are familiar with designing hierarchies of classes. We believe that modelling a hierarchie is much more intuitive than designing non trivial worth functions. Additionally the rejectment levels provide means to condense the set of states a QoS parameters might vary on the same level of preferences.

4.5. Mapping Preferences to QoS Parameters

Hierarchical grouping of client preferences by a refinement based approach enables the definition independent from the underlying QoS parameter representation. However, for a concrete realization a QoS management architecture must provide a mapping from client preferences to QoS parameters and an evaluation procedure such that the best matching QoS can be determined. The implementor of a QoS characteristic supports the QoS definition, i.e. the QoS parameter, and the realization through QoS mechanisms. Additionally, a uniform interface for evaluation and the base contract must be provided. Since all refined contracts contain less or the same dimensions of the base contract, a mapping from the base contract to the QoS parameter covers all refined contracts as well. The evaluation function simply takes a rejectment node (a refined contract) and

returns *true* or *false* depending on the system's ability to provide the corresponding QoS level.

The additional component for a QoS management architecture is a parser for the specifications which generates an interface with an evaluation function for each contract type. The QoS implementor has to implement this interface in order to link the preference representation with the underlying QoS parameters. A module which operates on the contract representation is linked to the service and can be invoked transparently because the dependencies to the underlying architecture is encapsulated by the evaluation interface. Choosing a benign representation of contracts is necessary, thus parsing can be accomplished with little runtime overhead and eases the implementation.

The next section briefly describes our QoS management architecture and the integration of client preferences.

5. MAQS: Management Architecture for Quality of Service

In the previous sections we have presented an approach to specify client preferences independent from the underlying QoS management architecture. These negotiation components are employed in our MAQS framework. MAQS (Management Architecture for Quality of Service) is an architecture for the integration of generic QoS management into object-oriented middleware. This section covers briefly the design and the status of our implementation. A more comprehensive overview is presented in Becker/Geihs (1998).

5.1. Design Goals

Object-oriented middleware like CORBA[8], DCOM[9] or Java/RMI[10] is popular for the design and implementation of distributed systems. However, current systems lack support for explicit QoS management. (The new CORBA messaging service[11] is a notable exception, and a clear indication that there is "a trend towards QoS"). We extend CORBA with QoS management. Using the already achieved abstractions for distributed object systems and enhancing them with QoS management leads to a clear design and to easy to use QoS extensions for clients and services. We adhere to the

[8] OMG (1995)

[9] Microsoft (1997)

[10] SunSoft (1998)

[11] OMG (1998)

guidelines of the Aspect Oriented Programming (AOP)[12] approach that enables the clear separation of QoS behaviour from functional behaviour. Hence, QoS implementations are reusable, existing services can be easily extended with QoS behaviour and non-QoS-enabled services and clients may be used as before. MAQS supports generic QoS management for multiple QoS categories. It allows the definition and implementation of arbitrary QoS characteristics.

5.2. Phases of QoS Integration

As stated before, QoS management has to be established in the three phases of service interaction:

1. Session Establishment: MAQS aims at generic QoS management. Therefore the QoS specification has to provide means for the definition of arbitrary QoS parameters. Negotiation has to be incorporated with client preferences.

2. Interaction: Monitoring the QoS provision and adapt to the better or worse is necessary to enable QoS enabled interactions in best-effort environments. Even in environments with resource reservation, policy based resource allocation can offer a greater flexibility to clients and services. The realization of policies makes adaptation necessary, e.g. detracting or assigning a resource leads to a worse/better QoS level, which has to be signalled to the involved client and service.

3. Accounting: If the interaction between client and service is done, the client has to be charged according to the delivered QoS level. This is important, since proper resource utilization depends on modest clients, which only request as much resources as they actual need. Pricing is a suitable mechanism to foster proper behaviour. Charging the service usage according to the delivered QoS is an additional advantage over flat rates, since lower quality leads to a lower price. Accounting information can be used to gather information needed to plan expansions and upgrades of the system.

5.2.1. Design and Implementation

During design and implementation of application clients and services QoS specifications are needed in order to support the QoS-enabling of clients and services. The QoS parameters are specified and assigned to services.

[12] Kiczales, G. et.al. (1997)

A QoS characteristic is realized through QoS mechanisms. From a software engineering point of view, the reuse of QoS implementations should be a major objective. This can be achieved by decoupling service and QoS implementation. However, QoS-enabled services need knowledge about the supported QoS and supporting behaviour. Our approach extends the Interface Definition Language of CORBA by QoS specifications and assignment to interfaces. QoS mechanisms can be implemented within an augmented Object Request Broker (ORB) at two layers - at the application layer or at the transport layer. Thus we gain a high flexibility for QoS mechanism implementations. The specification of client preferences is done using QML[13], an universal QoS specification language. QML constructs can be transmitted over the network as strings. Parsing QML and generating the hierarchy of contracts can be done with little effort, since the grammer is small and the LL(1) property holds. However, for the sake of efficiency we used a LALR-parser generator.

The QoS designer and implementor specifies the QoS parameter for a QoS characteristic and the base contract for client preferences in QML. The mapping from QML contracts to QoS parameters is implemented by the evalutation interface which is done by the QoS implementor as well. Thus applications can use client preferences represented in QML without knowledge of the internals of the mapping to QoS parameters.

5.2.2. Runtime

During runtime all three phases of service interaction have to be supported. At the negotiation phase, a client needs assistance to locate an appropriate service, which supports the desired QoS. This is done by using the trading service.

The negotiation between a client and a service relies on a specification of the client's preferences and a negotiation framework. The negotiation framework provides a set of standardized methods for the exchange of client preferences and negotiation results. The service implementor can rely on a module which parses QML contracts and evaluates client preferences.

Resource management and accounting is based on a resource service which provides allocation of resources and delivers these information to an accounting service. A monitoring service will issue notifications if the agreed QoS level is violated.

[13] Frølund, S./Koistinen, J. (1998)

5.3. Status Quo and Next Steps

In the first phase of our project we have focussed on the overall design of MAQS and the QoS specification. A prototype based on our CORBA implementation MICO[14] is available. In the next phase, we will design and implement the infrastructural services (trading, resource control, accounting, and monitoring). Since our research is embedded in a large interdisciplinary research program that investigates the effects of networking, accompanying projects with MAQS as distribution infrastructure are planned. Thus, feedback from users can be integrated at an early stage and other projects gain benefits from a QoS-enabled infrastructure.

5.4. Related Work

Integration of QoS management in distributed object systems is a research topic which has not been sufficiently answered yet. The OMG adresses QoS integration mainly in specific areas, like telecommunications[15] or the messaging service[16]. Though there have been efforts towards a more general approach[17] the actual QoS management of CORBA is done specifically for each domain. Negotiation and trading is done solely with the domain specific mechanisms. Explicit handling of client preferences with respect to QoS is not discussed or supported.

There are a number of single category QoS management architectures based on CORBA, like TAO[18] (real-time) or Elektra[19] (fault-tolerance). Basic interoperability with existing objects is done via CORBA. Extending the given set of QoS is not contemplated. Neither is the negotiation of QoS parameters based on client preferences.

QML is a QoS modelling language which can be used for QoS specification in modelling as well as for the generation of QoS-enabled services in distributed object systems. Koistinen and Seetharaman[20] describe the integration of worth functions in QML and a negotiation protocol to retrieve the best possible QoS regarding to a client's preference. However, the integration of client preferences with worth functions is not

---

[14] Römer, K./Puder, A. (1997)

[15] IONA Technologies, Lucent Technologies, and Siemens-Nixdorf (1997)

[16] OMG (1998)

[17] OMG (1997)

[18] Schmidt, D.C. et.al (1997)

[19] Maffeis, S. (1995)

[20] Koistinen, J./Seetharaman (1998)

an easy to use abstraction. Though the use of one modelling language, i.e. QML, reduces the complexity of a QoS management architecture, the collaboration of the overall system and components is still quite complex.

Another generic multi-category QoS management architecture for distributed object systems is QuO[21]. It uses the notion of a QoS region[22] which corresponds to a vector space mapping of QoS parameters. Clients can identify subregions which allows the specification of ranges a QoS parameter might vary.

An extension to OMG trading based on the comparison of distances in a vector space built from QoS properties is described by Linnhoff-Popien[23]. Customable metrics in the vector space allow different preferences to be specified. Though this approach lacks genericity it provides an abstraction of actual QoS parameters. However, it is not sufficient for negotiation and provides extensions for trading purposes only.

6. Conclusion and Outlook

We have accomplished a general specification and negotiation technique for client preferences for generic multi-category QoS management. Although the approach is targeted at our own QoS management architecture MAQS, it could also be used in other QoS management architectures. We have built our solution on top of QML, a universal QoS specification language, and we have added a mapping to our CORBA-based MAQS. The additional abstraction of QoS parameters in client preferences adds new expressiveness to clients. System dependent QoS parameters can be abstracted by the preference specification and translated by the mapping to the system QoS parameter representation.

Overall support for resource monitoring and management, as well as the integration of accounting based on resource consumption will be the next steps for our QoS management architecture.

Acknowledgements

We thank Oliver Wendt for an introduction to utility theory and preference relations. This work has been funded by the Deutsche Forschungsgemeinschaft (DFG) under SFB-403. The anonymous reviewers provided valuable comments.

[21] Zinky, J.A. et.al. (1997)

[22] Loyall, J.P. et.al. (1998)

[23] Linnhoff-Popien, C./Thießen, D. (1997)

References

Becker, C./Geihs, K. (1998): QoS as a Competitive Advantage for Distributed Object Systems, in Proceedings of EDOC'98, La Jolla, USA.

Frølund, S./Koistinen, J. (1998): Quality of Service Specifications in Distributed Object Systems, in Proceedings of COOTS'98, Santa Fee, USA.

International Organization for Standardization (ISO) (1995): Quality of Service – Basic Framework N9309

Kiczales, G. et.al. (1997): Aspect Oriented Programming, Technical Report SPL 97-08P97 10042, Xerox Palo Alto Research Center.

Koistinen, J./Seetharaman (1998): Worth-based Multi-Category Quality-of-Service Negotiation in Distributed Object Infrastructures, in Proceedings of EDOC'98, La Jolla, USA..

Linnhoff-Popien, C./Thießen, D. (1997): Integrating QoS Restrictions into the Process of Service Selection, in Proceedings of IWQoS'97, New York, USA.

Liskov, B./Wing, J. (1993): A New Definition of the Subtype Relation, in Proceedings of ECOOP'93, Kaiserslautern, Germany.

Loyall, J.P. et.al. (1998): Specifying and Measuring Quality of Service in Distributed Object Systems, in Proceedings of ISORC'98, Kyoto, Japan.

Maffeis, S. (1995): Adding Group Communication and Fault-Tolerance to CORBA, in Proceedings of COOTS'95.

Microsoft Corporation (1997): Component Object Model Introduction.

Object Management Group (OMG) (1995): The Common Object Request Broker: Architecture and Sepcification, Revision 2.0.

Object Management Group (OMG) (1997): Quality of Service (QoS) green paper.

Object Management Group (OMG) (1998): CORBA Messaging.

Römer, K./Puder, A.(1997) MICO, http://www.mico.org/

Schmidt, D.C. et.al. (1997): The Design of the TAO Real-Time Object Request Broker, in Computer Communications Journal.

SunSoft Inc (1998): Java RMI Documentation: http://www.sun.com/products/jdk/1.2/docs/guide/rmi/index.html

IONA Technologies, Lucent Technologies and Siemens-Nixdorf (1997): Control and Management of Audio/Video Streams – RFP Submission.

Zinky, J.A. et.al. (1997): Architectural Support for Quality of Service for CORBA Objects, in Theory and Practice of Object Systems, Vol 3(1), pp 55-73.