

Workshop on Quality of Service in Distributed Object Systems

Christian Becker

John Zinky

Abstract. The suitability of the object-oriented programming model for implementing distributed systems has lead to middleware platforms, such as CORBA, DCOM, and Java/RMI. Originally, these middleware systems aimed at distribution transparency for application programmers. However, distributed systems are exposed to system issues, such as dynamic performance changes or partial errors, which prevent complete distribution transparency. Quality of Service (QoS) management addresses these system issues. The goal is to add QoS management to the interactions between clients and services. Support for QoS management in distributed object systems is a hot topic of current research which poses a number of open questions: How is QoS integrated with the object model that emphasizes encapsulation and information hiding? Can one build generic support frameworks for multiple QoS categories, in contrast to specialized, single category systems, such as TAO, Electra, Eternal, DOORS among others. Can QoS be viewed as an aspect in the sense of Aspect Oriented Programming (AOP) or are other classifications more appropriate? This ECOOP-workshop has discussed the open questions and aimed at a summary of the state of the art in the field. The workshop stimulated discussions about how next generation QoS management facilities can be built into object infrastructures.

1 Organizers

Christian Becker

FB Informatik
Verteilte Systeme und Betriebssysteme
Robert-Mayer-Str. 11-15
60325 Frankfurt am Main
Germany
becker@vsb.informatik.uni-frankfurt.de

John Zinky

BBN Technologies
Part of Verizon
10 Moulton (6/3d)
Cambridge, MA 02138
USA
jzinky@bbn.com

Workshop Homepage

<http://www.vsb.cs.uni-frankfurt.de/misc/QoSDDS/>

2 Workshop Objective

Distributed Object middleware has emerged to solve the problems of distributed resources and heterogeneity in operating systems and languages. This middleware makes programming distributed applications easier. The programmer only

has to deal with one set of standard interfaces which hide platform specific dependencies. The middleware's runtime uses standard protocols and formats so that subsystems created by different programmers with different tools can have the resulting systems interoperate. Finally middleware provides a high-level abstraction, so less new code has to be written and old code can be reused.

Several middleware standards have emerged. Microsoft's DCOM is a distributed version of the COM application interface and is used to connect Microsoft-based application together. OMG's CORBA is a standards based middleware that addresses the heterogeneity issue. Dozens of CORBA implementations exist for many different languages and operating systems. Also, commercial and free-ware version of CORBA exist. Sun's Java RMI uses the Java virtual machine for it's interoperability. Finally, web based object systems, such as SOAP, are emerging. Each of these Distributed object systems have their own niche and will co-exist for the immediate future.

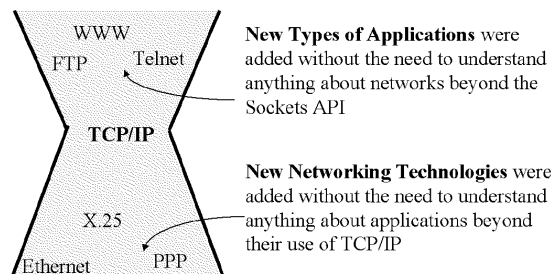


Fig. 1. The Simple Abstraction of TCP/IP Interfaces Allowed Networks and Applications to Evolve Independently

While the heterogeneity problem has been addressed by middleware strategies, wide-area (WAN) distributed applications are still hard to build and maintain. One of the problems is handling the high variability in the underlying components. First, WAN-based resources are more dynamic, unpredictable and unreliable. Application programmers are not used to writing code where an object can disappear between calls or may take an arbitrary amount of time to complete a call. Second, WAN-based applications have a wider range of end-users, each with different QoS requirements. The programmer is burdened with handling a wide range of usage patterns and requirements, which is hard to meet with just one implementation. Third, many resource allocation decisions can not be made until run time. Control of resources is an administrative matter in the

user's environment and can't be successfully managed exclusively by the programmer. Finally, many algorithms exist for accomplishing the same task, but which use different mixes of resources. Programmers are forced to use a general algorithm that works just adequately over a wide range of situations, instead of a custom algorithm that works well, but for a limit situation.

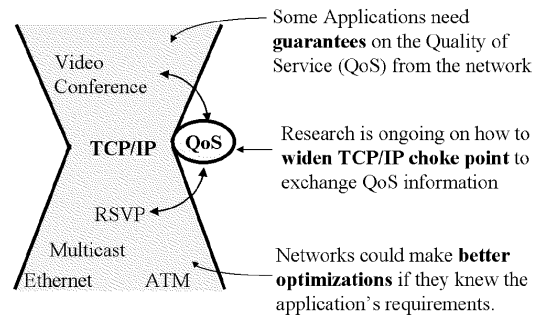


Fig. 2. Applications and Networks can be Implemented More Efficiently, If More Information is Exchanged Across the Interface Boundary

Some lessons can be learned from the successful deployment of the Internet. Dave Clark from MIT (circa 1996) has an explanation for why TCP/IP was so successful. His claim is that the socket-layer abstraction and the TCP/IP protocol stack allowed networks and applications to grow independently (Figure 1). New types of applications were added to the Internet without having to change the networking infrastructure. For example, the World Wide Web was a completely unanticipated application which was built on top of TCP. The network also evolved without changing the applications. For example, the networking communities introduced new technologies, such frame relay and ATM, without changing any upper-layer protocols. Distributed object middleware has a similar role of allowing applications to evolve independently from the underlying system mechanisms.

But the best-effort model for network communications does not work in all situations. Also, the guaranteed resource model of the phone-networks is too rigid for most computer-based applications. In hindsight, a spectrum of communication services is needed. Applications and networks could be implemented more efficiently, if more information were available between them (Figure 2). Thus, Quality of Service (QoS) has become the language for moving additional information between applications and networks. QoS API's defines how each should behave. Applications, such as video conferencing, need QoS guarantees from the

network. Also, the network needs restrictions on the traffic patterns generated by the applications. Research in QoS has widened the TCP/IP choke point to include exchanging QoS information. What was once a simple interface has exploded into a myriad of partial solutions which are still in flux after almost ten years. Because of their complexity and volatility, these network-based QoS interfaces need be hidden from the application developer. A similar situation is happening in operating systems and database research, which are also adding the notions of QoS to help manage CPU and storage resources.

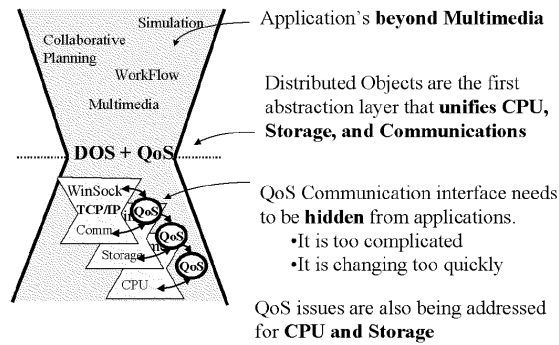


Fig. 3. Distributed Object Systems with QoS Extensions is a New Abstraction Layer on which to Build Distributed Applications

From the application's prospective, the key to managing QoS is making tradeoffs among resources. For example, knowing only about network resources is not enough information to decide whether a transfer should be compressed or not. Compression uses more CPU and memory resources in order to reduce consumption of network resources. The socket abstraction presented by the TCP/IP layer has no notion for CPU and memory resources, hence it is not powerful enough to manage QoS at the application layer. Distributed objects is the first abstraction layer that unifies CPU, storage and communications resources (Figure 3). The object's state uses storage resources; the object's methods use CPU resources; and the client/objects interactions use communications resources.

Classic distributed object middleware tries to hide the underlying infrastructure from the application developer in the same way that TCP/IP hides the network infrastructure from the socket-layer. Like TCP, the black-box approach is not an adequate API to develop adaptive applications, so distributed object middleware must be extended to manage QoS. But unlike TCP/IP, distributed objects can unify the tradeoffs between the basic resources. Distributed object middleware with QoS extensions will be the new choke-point on which to develop

distributed applications. This middleware will allow applications to develop independently of the new QoS mechanisms being created to manage resources.

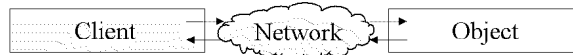
3 Classification of QoS Middleware

Although adding QoS to distributed object middleware is necessary, the state of the art is still at the experimental stage. Currently, many different schemes have been proposed and implemented as different groups have addressed QoS issues in their distributed object systems. This workshop looked at participant's existing distributed object systems and tried to dissect them to see how they addressed QoS issues. The hope was to come up with a list of questions that can differentiate and categorize the different QoS approaches. The following questions were refined as part of the workshop:

- QoS is well-controlled Client and Object are at the same location



- QoS is unruly across the network



- How can we resolve this?

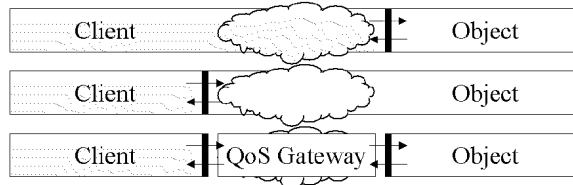


Fig. 4. QoS Properties are at the Boundary Between the Client and Object

What Kind of QoS is being handled?

Traditionally in the networking community, QoS has been associated with meeting guaranteed performance requirements for multi-media applications. Recently in the DARPA Quorum program and QuOIN project, the notion of QoS has been extended to include other system properties, such as real-time, dependability, security, and resource management. In addition, the notion of QoS has been extended to be "end-to-end" which goes beyond just the network to include the

hosts and storage resources. Research has focused on managing only one kind of QoS at time, such as a real-time ORB or dependable ORB. Future research will attempt to combine multiple QoS properties, such as a secure real-time ORB. But sometimes the QoS properties conflict, which forces trade-offs between QoS properties. For example, overhead of security operations may preclude meeting real-time deadlines. Each distributed object system deals with a specific set of QoS properties, such as the TAO from Washington University at Saint Louis concentrates on real-time properties.

Who is responsible for the QoS?

Client and objects do not have QoS properties themselves, but the interaction between them does. When clients and objects are in the same process on the same host, QoS between them is well controlled and basically can be considered "infinitely good". But when a network connection separates the client from the object, QoS becomes an issue (Figure 4). Current middleware, such as CORBA, does not manage network resources. This works fine as long as network resources are not the bottleneck, such as when using a high-speed LAN. But, when a resource is scarce, something has to be responsible for managing it. The burden can fall on the client, for example a smart browser accessing a web page. The burden can fall on the object, such as a load balancing server. Finally, the burden can fall on an intermediary, such as a QoS gateway or firewall. Each distributed object system offers mechanisms for supporting clients, objects or intermediaries.

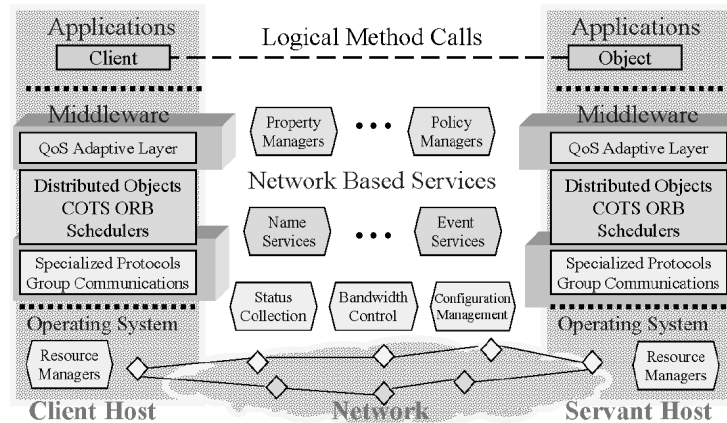


Fig. 5. Current ORBS must be Extended to Handle QoS Interfaces and Control

Where does QoS adaptability occur?

Mechanisms for managing QoS fall into distinct layers (Figure 5). Underlying resources could manage QoS for themselves. But the QoS abstractions are at the resource-level and are hard for the application developer to use directly. The ORB can be specialized to manage a specific type of QoS. The QoS API is defined by the ORB and cannot be changed by the application developer. A wrapper-layer can be added above the ORB to add QoS support which the application developer can manipulate. Also, QoS services can be implemented as an overseer function in-between the client and object. Each distributed object system modifies layers in order to support QoS.

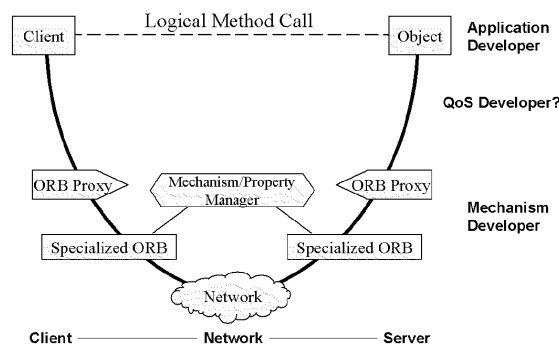


Fig. 6. QoS Management has Different Roles

What roles are supported?

Applications are not developed by a single programmer. Programming tasks are divided between roles which perform a specific kind of development (Figure 6). Application developers want to create an application at the highest level possible. Application developers can be further divided in object developers who create a library of general purpose objects and client developers who hook the general objects together to make a specific applications. At the other end of the spectrum, mechanism developer are adding QoS management techniques into the underlying system. The QoS developer is a new role for creating libraries of adaptive schemes, which can be used by the object designer to create adaptive objects. These roles need radically different skill sets. Each distributed object system supports one or more of these roles.

In what Time horizons does adaptability occur?

The object life-cycle has many time epochs in which adaptability can occur. At compile time, the marshaling is fixed and efficient stubs can be generated. At configuration time the application can be installed on the best servers. At connection time the type of adaptability can be bound. Finally at invocation time, the call can be routed to the best server. Each distributed object system supports adaptation for a specific set of time horizons

How is QoS adaptability reused?

The most important service a distributed object system can support is reuse of adaptive mechanisms. These mechanisms can be hardwired into an underlying service, such as the RSVP protocol. Others can be in reusable libraries or templates. Or the adaption can be supported by code-generators or compilers. Each distributed object system offer mechanisms for roles that can create adaptive mechanisms which can be reused.

4 Summary of Contributions

The organizer were delighted by the responses to the call for papers. The quantity and quality of the submissions exceeded our expectations. The review process selected papers not only according to their technical quality but also with respect to their potential to foster discussion. The contributions were classified along the end-to-end provisioning of Quality of Service in distributed object systems. The intention was to group the presentations according to their responsibility for QoS provisioning and to ensure that each presentation focussed on a distinct topic. A summary of each session appears below.

4.1 QoS Support in and below ORBs

The theme of this session was the provisioning of QoS below ORBs, i.e., operating system and network-related QoS provisioning. We thought that an emphasis should be given to the impact of ORBs residing on QoS enabled platforms, hence the second topic in this session dealt with the integration of QoS mechanisms in ORBs. In total, four papers were presented in this session.

Two of the papers in this session dealt with the structure of communication in middleware architectures. Both approaches use generators to configure the protocols according the required QoS provisioning. Though one approach explicitly follows the principles of Aspect Oriented Programming (AMIDST [1]) the other approach (Jung and Biersack [3]) could be classified in these terms as well. A common conclusion holds that relying on TCP/IP – as most middleware architectures do – is not appropriate for QoS provisioning. Configuring the middleware is done in each case through generators. While the AOP-based approach uses information from the application-centered QoS requirements to configure

appropriate transport services in the ORB, the non-AOP approach allows customizable protocols through use of a generator. However, specifying QoS in the latter is still an open issue. Both address the separation of concerns with respect to QoS and application logic.

The other two papers addressed QoS from a more platform-oriented perspective. The design of a QoS-aware operating system kernel (Off++ [4]) and the integration of active networks in a CORBA-environment (IST-FAIN [2]) show the connection of the protocol-oriented approaches to the underlying platform. The presented operating system kernel exposes QoS-related entities to application developers. A discovery protocol allows exploration of available resources. The kernel itself allows remote access to network related entities and their allocation. This work aims at the improvement of underlying platforms for the QoS-aware middleware above. In contrast, the integration of active networks and distributed processing environments requires an end-to-end view. Both the connection between application objects and their QoS requirements, and the QoS provisioning on the network are done through a modified bind process between client and server. The bind process can access the underlying active network through distinct abstractions, e.g., router-access interfaces. The communication protocols in the ORB can then rely on the active network's QoS support.

4.2 QoS Specification

Though the majority of presentations were concerned with QoS frameworks based on object-oriented middleware, we selected two submissions related to the specification of QoS requirements from an application designer's point of view. We viewed this as an ideal supplement to the QoS frameworks session since most QoS frameworks provide specifications for QoS issues as well.

The specification-related contributions cover two areas of QoS specifications. The first area-involves the specification of the QoS requirements for multimedia applications in the OMODIS system [5]. In order to allow Lecture-on-Demand (LoD) applications, the specification of desired application-specific QoS requirements of data stored in a multimedia database, hierarchical contracts are introduced. QoS contracts model the user requirements in a multimedia presentation. Hierarchies offer convenience for complex presentations. They allow the specification of global QoS requirements, which can be refined for distinct parts of the presentation. The proposed contract hierarchies allow negotiation of a complete multimedia presentation. Adaptation can be established in a fine granular manner, since only violated sub-contracts have to be renegotiated.

The deployment of QoS dependent applications in the telecommunications domain is the focus of the EURESCOM Project 924 [6]. QoS requirements exist between components and are known up front. Though this restriction seems to ease the solution, the contribution motivates the lack of deployment support of current middleware systems as well as the need for QoS awareness in such services. The QoS aspects of concern are object locations, resource assignments, duplication and migration of objects. The proposed specification framework consists of UML specifications for the functional aspects and a distinct language for

the QoS constraints. These specifications are woven into an XML specification that is used to generate an installation map, scripts for the application setup and runtime constraints.

Both specification contributions illustrate the necessity of complex specifications when real-world requirements are addressed. Even the sole addressing of multimedia presentations requires much more support from QoS specification than the “classic” bandwidth and throughput issues, which are mostly concerned with a distinct channel between client and service.

4.3 QoS Frameworks above the ORB

This session – with six presented contributions and even more submitted – represented the major part of the workshop. Although QoS frameworks typically consist of specifications as well as embedded QoS mechanisms below the ORB, this session was dedicated to contributors describing their work in a larger context.

The contributions can be classified as generic multi-category QoS frameworks and single category multimedia supporting frameworks. While one of the multimedia approaches (Djinn [8]) uses middleware only to control the interactions in a multimedia system and provide a two layer abstraction of the components involved, the other approaches all rely on modified ORBs.

The modified ORBs all address the necessity of integrating customized protocols. As mentioned in the “QoS below and in the ORB” session currently middleware relies mostly on TCP/IP, which is not sufficient for most QoS requirements. Hence, each approach provides distinct dispositions for the integration of application-specific transport protocols.

One approach uses QML – a QoS specification language from HP Labs, Palo Alto – to describe the QoS requirements (Dittrich, Solariski [9]). The QML specification is mapped to an augmented ORB, which provides means for protocol integration and an extended binding interface through which application objects can access the appropriate QoS transport. Reliability, scalability and availability are the supported QoS categories.

The specification of QoS requirements by nested contracts from the QoS specification session is mapped to an augmented ORB in the MULTE project [10]. This ORB provides a layered architecture consisting of application specific mechanisms, QoS binding related logic, and QoS mediation to underlying resources. The three layers offer a wide variety of different QoS implementations with flexible signaling mechanisms. It seems likely that this architecture is suitable for other specification mappings as well.

One of the generic approaches introduces a QoS meta model for the specification of QoS contracts using XML and the Meta Object Facility (MOF) (Daniel, Traverson, Vignes [7]). The QoS contracts, specified in a distinct language (QoS Description Language (QDL)), are mapped to QDL. The underlying QoS management framework is responsible for enforcement of the QoS contracts (guaranty, observation, negotiation and composition). The authors claim to provide QoS management independently of the application domain.

The last two contributions present an AOP-based approach. However, both differ in the choice of aspect languages. While MAQS [12] uses an extension to OMG-IDL in order to describe the QoS and application specific interfaces, QuO [11] offers a variety of specialized description languages. Both approaches offer the integration of application-specific QoS mechanisms at multiple layers in and around the ORB. The AOP-based approach separates the QoS related behavior from the application behavior. Since programmers are familiar with IDL-compilers for building applications using middleware platforms, enhancing the IDL-compiler or an additional phase of IDL-compilation is a good solution for weaving of QoS aspects.

4.4 QoS enabled Applications

This session has been dedicated to QoS provisioning in object systems from an application developers view. We hoped for several examples of QoS integration in object systems. Unfortunately, there were some cancelations and thus only one participant presented in this session. DVECOM project [13] deals with end user QoS requirements in a distributed virtual environment for Computer Supported Collaborative Work (CSCW). The QoS characteristics of interest are related to synchronization and real-time aspects of service provisioning.

5 Conclusions and Recommendations

The most important conclusion is perhaps the most obvious, that QoS must concern more than the classical consideration of bandwidth, delay and jitter provisioning. QoS must concern security, dependability and other performance related aspects as well.

The integration of QoS mechanisms in middleware infrastructures is addressed with similar integration points and QoS provisioning responsibilities. The lack of standardization is quite evident in that area, since all presented approaches state the same needs and offer similar solutions.

In contrast to the mechanism integration in the ORB, which is rather agreed upon, the specification of QoS requirements and the integration of QoS provisioning into the application are handled quite differently. The generic approaches featured aspect-oriented programming based on weaving of aspect languages. However, the degree of information factored out into the aspect languages differs. As well, the specification of the non-AOP approaches differ greatly. The necessity to specify QoS parameters that reflect the state of the system with respect to a certain QoS characteristic is addressed quite differently among the presented architectures. Some solutions focus on the negotiation of QoS agreements and offer specifications tailored to this purpose, while others only represent the state and add an additional specification for negotiation. We view this area as an important field for further research.

Middleware integrates QoS management from applications to the underlying platform. It is crucial to provide clear mappings between QoS specifications and

the corresponding behavior in order to establish the end-to-end provisioning of a QoS characteristic. However, the integration of QoS behavior in applications is handled differently by each approach. It is still unclear whether a generic framework can cover all domains of QoS management. The requirements on the underlying platform that arise from QoS provisioning by middleware, also warrants further study.

6 Lessons Learned by the Organizers

The workshop attracted a variety of researchers from the domain of QoS management in distributed object systems. This enabled a selection of presentations not only driven by the quality of the submitted abstracts, but also governed by the workshop program. As a result the workshop covered not only the impact of QoS integration on middleware architectures but also adjacent areas, i.e., operating system, network, applications, and QoS specification. We consider this as an ideal base when discussing middleware. However, a typical presentation-questions-panel organization might not be the best to foster discussions among the participants.

Although the variety of approaches made it difficult to cut down the program to even fewer presentations, this would have allowed more and deeper discussions. Nevertheless, bringing together researchers from all QoS related layers proved useful, since the exchange of requirements and experiences can be helpful. In order to allow this exchange happen, the workshop size should not grow beyond 15 presentations.

We believe the workshop meets a need that is not addressed by other workshops, e.g. Middleware, IWQoS, Reflective Middleware. These venues are either too broad, e.g. covering all middleware or QoS aspects with a resulting high number of participants, or too specialized, e.g. Reflective Middleware.

7 Participants

Mehmet Aksit
aksit@cs.utwente.nl
University of Twente, Netherlands

Ali Noutash
Ali.Noutash@oti.com
Object Technology International, Ottawa, Canada

Thomas Becker
becker@fokus.gmd.de
GMD-Fokus Berlin, Germany

Matthias Jung
Mathhias.Jung@eurecom.fr
Eurecom, Sophia Antipolis, France

Francisco J. Ballesteros
nemo@gsync.escet.urjc.es
Rey Juan Carlos University of Madrid, Spain

Denise Ecklund
denise@uniko.no
UniK, the Center for Technology at Kjeller, Norway

Thomas Unterschütz
Thomas.Unterschuetz@telekom.de
T-Nova, Darmstadt, Germany

Jerome Daniel
Jerome.Daniel@der.edf.fr
EDF, Clamart, France

Hani Naguib
hanin@dcs.qmw.ac.uk
Queen Mary and Westfield College, GMD-Fokus, Berlin, Germany
London, UK

Jürgen Dittrich
dittrich@fokus.gmd.de

Marcin Solarski
solarski@fokus.gmd.de
GMD-Fokus, Berlin, Germany

Tom Kristensen
tomkri@ifi.uio.no
University of Oslo, Norway

Zied Choukair
zied.choukair@enst-bretagne.fr
ENST Breteagne, France

Lodewijk Bergmans
bergmans@cs.utwente.nl
University of Twente, Netherlands

References

1. Mehmet Aksit, Ali Noutash, Marten van Sinderen and Lodewijk Bergmans *QoS Provision in CORBA by Introducing Reflective Aspect-Oriented Transport Layer*
2. Hui Guo and Thomas Becker *Active Distributed Processing Environment*
3. Matthias Jung and Ernst W. Biersack *QoS for Distributed Objects by Generating Tailored Protocols*
4. Francisco J. Ballesteros, Fabio Kon, and Roy Campbell *QoS in the Off++ kernel*
5. Denise Ecklund, Vera Goebel, and Thomas Plagemann *Nested QoS Contracts with Inherited QoS Parameters*
6. Thomas Unterschütz and Veit Vogel *Distribution and Configuration Support for Distributed PNO Applications*
7. Jerome Daniel, Bruno Traverson and Sylvie Vignes *A generic QoS management framework for distributed environments*
8. Hani Naguib, George Coulouris, Heather Liddel and Scott Mitchell *Managing QoS in a component-based framework*
9. Jürgen Dietrich and Marcin Solarski *QoS Enhanced Middleware*
10. T. Plagemann, F. Eliassen, B. Hafskjold, T. Kristensen, R. H. Macdonald and H. O. Rafaelsen *Managing Cross-Cutting QoS Issues in MULTE Middleware*
11. John Zinky *Quality Objects Project*
12. Christian Becker *Management Architecture for QoS (MAQS)*
13. Zied Choukair and Magnus Hellström *Implementation and test of QoS Management in a Distributed Virtual Environment Model for CSCW*