A framework for QoS-enabled middleware

Cosmina Ivan

Department of Computer Science, Faculty of Automation and Computer Science

Technical University of Cluj, Romania

e-mail : cosmina.ivan@cs.utcluj.ro

Abstract. Developers of distributed multimedia applications face a diversity of multimedia formats, streaming platforms and streaming protocols; furthermore, support for end-to-end Quality-of-Service (QoS) is a crucial factor for the development of future *distributed multimedia and real time systems*. Middleware is gaining wide acceptance as a generic software infrastructure for distributed applications, a growing number of applications are designed and implemented as a set of collaborating objects using object middleware, such as CORBA, EJB and (D)COM(+), as a software infrastructure that facilitates distribution transparent interactions. However, quality aspects of interactions between objects cannot be specified nor enforced by current object middleware, resulting only in a *best-effort QoS* support in middleware. Next generation object middleware should offer abstractions for management and control of the system level QoS mechanisms, while maintaining the advantages of the distribution transparencies, these abstractions should take into account that new interfaces to OS resources and new network protocols are expected to appear as the result of ongoing research efforts, in addition, a changing run-time environment must be handled. The paper discusses the design and implementation of a QoS-enabled middleware platform with QoS guarantees. Properties of content delivered are represented using a generic content representation model described using the OMG Meta Object Facility (MOF) model. The integration of the QoS support, content representation and content delivery framework results in a *QoS-enabled middleware* which is representation, location and QoS transparent. This paper presents aspects of the framework for controlling QoS at middleware level implemented as a provisioning QoS service, suporting this new paradigm of QoS-enabled middleware .

I. INTRODUCTION

Middleware provides distributed objects with a software infrastructure that offers a set of well-known distribution transparencies. These transparencies enable the rapid introduction of applications for heterogeneous, distributed systems. However, to support guaranteed Quality of Service (QoS) system-specific QoS mechanisms need to be controlled. Allowing applications to directly access and control these mechanisms would negatively impact the distribution transparencies offered by the middleware layer and would reduce the portability and interoperability of distributed object applications. To avoid this, *next generation object middleware should offer abstractions for management and control of the system level QoS mechanisms*, while maintaining the advantages of the distribution tranparencies.The challenge for next-generation middleware is to support application-level QoS requirements, coherent mapping on low level mechanisms,while maintaining the advantages of the distribution transparencies.

This paper is organised as follows. Section 2 describes the QoS concept in open distributed systems. Section 3 gives an overview of the requirements for a middleware-based software infrastructure that offers QoS support to distributed objects and a survey of existing frameworks Section 4 presents our solution in the form of an infrastructure service for QoS provisioning. With a short evaluation of this framework based on the implementation section 5 completes with conclusions and further developments.

II. A CONCEPTUAL FRAMEWORK

Provisioning of QoS usually involves a common understanding between the two or more parties about the quality characteristics of the service, these parties can be end-users or software components. The generic concepts are based on the ISO/IEC QoS Framework, and the provisioning model defines as main entities in the system the service provider and the service user. Often the user requirements are expressed as subjective requirements whereas the service provider needs objective requirements in order to handle them, therefore the user requirements must be translated into specific QoS parameters expressed in terms of QoS characteristics of the Service provider. The QoS provisioning model should enable entities to express their quality requirements. The relevant concepts for a QoS provisioning model [2] are defined as follows:

- *QoS characteristic:* a quantifiable aspect of QoS, which is defined independently of the means by which it is represented or controlled.

- *QoS requirement :* QoS information that express part or all of a requirement to manage one or more QoS characteristics, when conveyed between service user and provider a QoS requirement is expressed in terms of QoS parameters.

The QoS characteristics of the service provider are determined by the QoS management functions

- *QoS management function* define a functions specifically designed with the objective to meet QoS requirements.
- *QoS mechanism* define a specific mechanism that may use QoS parameters or QoS context possibly in conjunction with other QoS mechanisms in order to support establishment, monitoring, maintenance, control or enquiry of QoS.

QoS management architectures provide a coherent set of abstractions [2,9] and components in order to enable applications for QoS handling.A QoS framework combines interfaces and mechanisms that support the development, implementation and operation of QoS enabled applications, and also infrastructure services such as for the negotiation of QoS agreements and monitoring them.

Thinking of QoS provision as a client-server relationship means the interaction between them must be augmented with QoS specific behaviour. The integration of QoS provision in middleware must address the following points:

- *QoS specification* : QoS parameters have to be specificed along with operations for a dinstinct mechanism in order to be customized, configures and monitored.
- *QoS mechanism integration* : the specification of QoS parameters and the operation of the related QoS mechanism is not enaugh , the provision mechanism need to be integrated with specific mechanisms which can ensure end-to end QoS requirements.
- *QoS binding* in order to attribute the interactions between a client and the service with a distinct QoS an assignmenmt of a QoS characteristic to the client-server relationship has to be established
- *QoS adaptation* : the possible level of QoS characteristics depends on the resource availability in the system , each QoS agreement has to be negotiated independently, and varying resource availability should be addressed through adaptation

Functions that realise QoS support in a distributed processing environment and their positioning in an open distributed system are designed mainly based on the following principles: *the separation principle* which states that transfer , control and management of data are distinct activities and *the integration principle* states that QoS must be configuable, predictable and maintainable over all architectural layers to meet end to end QoS, both principles derived from multimedia and broadband networks.

There are various requirements on QoS design concepts but the most important ones are : extensibility, composability, and a verifable and suitable run time representation. Following those requierements , we propose a layered architecture to structure the problem space and position the functions that provide QoS support in an open distributed system. In this architecture, three functional layers are distinguished, each with distinct responsibilities, offering services to adjacent layers on top and using services of adjacent layers below. Orthogonal to the layers, *three planes* are identified: data transfer, control and management, as for ATM architecture.

At the middleware layer, the responsibilities of the planes are as follows:

- the *data transfer plane* consists of the functions that provide the 'traditional', i.e. non-QoS related, distribution transparencies (in case of CORBA (Portable Object Adapter , the GIOP protocol engine or a CDR encoder.)
- the *control plane* is responsible for controlling the QoS mechanisms and monitoring the achieved QoS level to ensure adequate end-to-end quality of service levels, with the scope limited to a single association between objects.
- the *management plane* contains functions for long term monitoring,

The architectural framework presented in this paper has been developed to be flexible and re-usable and the main benefit that it allows us to combine and balance solutions for the control of multiple QoS characteristics.

QoS specification based on meta-modelling

The OMG Meta-Object Facility (MOF) [5] is a generic framework to define and represent meta-data. Meta-data denotes any data that describes properties of other data. In the MOF, a meta-model refers to a collection of meta-data. A MOF (meta-) model is an abstract language that can express this collection of (meta-) data. The MOF allows the definition of meta-models that are potentially domain independent and architecture neutral Modelling data recursively as meta-data leads to a potentially infinite number of meta-levels.

A QoS meta model [6] consists of two parts: a part that defines QoS *contract types* and a part that defines QoS *contracts*. The latter part also relates the contract to its contract type.

A QoS contract consists of a set of constraints on QoS dimensions. A contract must be associated with a contract type. A contract is only valid when it defines constraints for dimensions that have been defined for its associated contract type.The container-contained pattern,a variation on the composite design pattern [9] is used to model the relationship between a contract and a single constraint and in a similar way the relation between a multi-constraint and a statistical constraint. The meta-model constraints are formulated in such a way that a contract may contain single constraints and multi-constraints, while a multi-constraint may contain statistical constraints,the classes and their relations are represented as a UML class diagram.

III. QoS MIDDLEWARE ARCHITECTURES

The original motivation for introducing middleware platforms has been to facilitate the development of distributed applications, by providing a collection of general-purpose facilities to the application designers.

Currently, commercially available middleware platforms, such as those based on CORBA, are still limited to the support of best-effort Quality of Service (QoS) to applications.

Ideally, a middleware platform should be capable of supporting a multitude of different types of applications with different QoS requirements, making use of different types of communication and computing resources, and adapting to changes, e.g., in the application environment and in the available resources[1].

Middleware provides distributed objects with a software infrastructure that offers a set of well-known distribution transparencies. These transparencies enable the rapid introduction of applications for heterogeneous, distributed systems. However, to support guaranteed Quality of Service (QoS) system-specific QoS mechanisms need to be controlled. Accessing the low-level mechanisms directly by applications crosscuts the transparency offered by the middleware and limits portability and interoperability.

The middleware layer is a natural place for *brokering between QoS requirements of applications and the QoS capabilities of operating systems and networks.* The aim of a QoS-enabled middleware therefore is to provision QoS of applications in a heterogeneous distributed environment. Such a system has to deal with the diversity of low-level resource management mechanisms and the dynamic behaviour of the environment. The following requirements have been identified and are used as constraints on the design of our QoS provisioning service:

- *applications should be able to specify their QoS requirements using high-level QoS concepts.*
- *the software infrastructure should be modular and easily extensible with new interfaces to system level QoS mechanisms*, specifically, it should be possible to configure into the middleware components handling the control of different resource management mechanisms dynamically. Consequently, QoS control mechanisms are expected to offer standardized interfaces to the middleware, including reflective interfaces for run-time discovery.
- *the software infrastructure should allow adaptive QoS support.* In distributed environments the system behaviour is dynamic and only partially predictable, this requires adaptation that can be initiated both at the application and at the system level. Adaptation at system level on the other hand occurs when the availability of system resources drops, due to failure, system reconfiguration, increased user load or other non-predictable factors. Again, the middleware should re-allocate resources, and if possible, this should be completely transparent for applications.
- *the software architecture should support policies for the QoS negotiation* between client and server sides, and balancing and trading functions when resources are interchangeable.

QoS-enabled middleware is being developed in several projects, with different focuses. We describe here only those systems that enable applications to specify their QoS requirements using *high-level language concepts and realise resource adaptation.* Three main QoS-aware middleware groups can be identified: general purpose middleware, real-time middleware and multimedia middleware.QuO is a CORBA based framework for configuring distributed applications with QoS requirements[8]. It comes with a suite of description languages that allow applications to specify the interdependencies between QoS properties and system objects, thereby configuring the adaptive behaviour of the underlying system, but allows QoS mechanisms to be added at design time. MULTE-ORB is another QoS-aware middleware that supports configurable multiple bindings [7]. A QoS requirement is specified per binding, together with policies for negotiating QoS and for performing connection management. But the QoS configuration and management system is however ChorusOS and SunOS specific.OMG's Real-Time CORBA (RT-CORBA) specification is targeted at real-time distributed systems. Applications specify policies that guide selection and configuration of protocols and RT-CORBA supports explicit binding in order to validate the QoS properties of bindings. After binding time, however, protocols may not be reconfigured. TAO is a real-time CORBA ORB implementation targeted at hard real-time systems.

IV. DESIGN AND IMPLEMENTATION

A. Design elements

The adaptable QoS service is a *control plane service*, because its actions are limited to a single association between a client and a server object, acting as a broker between the application level QoS requirements and the available QoS mechanisms of the distributed resource platform . The service is a broker and controller for QoS agreements and the framework can be implemented as a CORBA service designed to use standard CORBA extension hooks, based on the Portable Object Adapter , the Portable Interceptor and the Open Communications Interface [6],[7].

A *control system* consists of a *controlled system* in combination with a *controller [Fig.1]*. The interactions between the controlled system and the controller consist of monitoring and manipulation performed by the controller on the controlled system. In QoS-enabled middleware, the 'controlled system' is the middleware functionality responsible for the support of interactions between application objects, while the 'controller' provides QoS control. Here, the environment represents the operational context of the middleware, which consists of application objects with QoS requirements and QoS offers.

The middleware platform encapsulates the computing and communication resources at each individual processing node, which may be manipulated in order to maintain the agreed QoS. Figure 2 shows the specialisation of a generic control model for controlling the QoS provided by a middleware

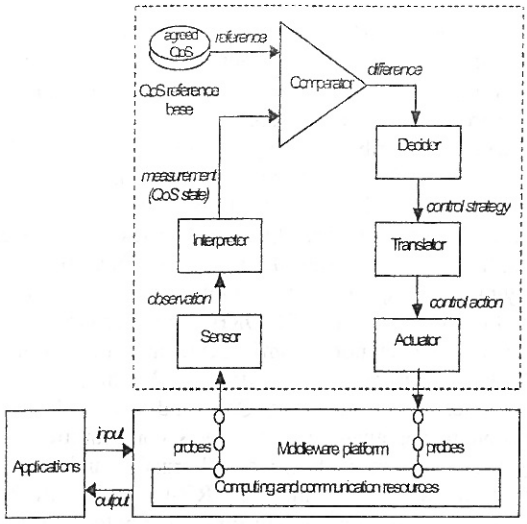

Fig.1. Conceptual QoS architecture

Since the service uses standardized ORB extension hooks, it can work with any standard ORB implementation that implements these extension hooks. On the server side, the POA is extended with a dedicated ServantLocator and a Negotiator object for managing servants with an offered QoS, and on the client side QAPS provides a QoSRepository (QR) interface for managing QoS requirements of clients.

The lifecycle of bindings controlled by the service revolves around the QoS level offered *(Qoffered)* by a server object, the QoS level required *(Qrequired)* by a client object and the agreed QoS level *(Qagreed)*. The purpose of the service is to control the resources in such a way that some *Qagreed* is negotiated and maintained for the lifetime of the binding. This agreed QoS is the result of a matchmaking process between the offered QoS of the server object and the required QoS of the client. The lifecycle phases are inform, negotiate, establish, operate and release .

During the negotiate phase, the service initiates a negotiation procedure between the client, the server and the resource platform to see if an agreement can be reached. A successful negotiation results in a *Qagreed* that is then associated with the binding, and resources are reserved for the binding. During the establish phase, the service assigns the resources that have been reserved during the previous phase, so other bindings cannot claim these resources. Once sufficient resources have been assigned to the binding, *Qagreed* must be maintained, this means correcting drifting quality levels, for example, by re-allocating system resources or, in case it is not possible, by informing applications to take appropriate actions. Applications can subsequently decide to lower their *Qrequired* and request a re-negotiation, or end their binding, this is the operate phase and finally, when the client does not further need the binding (this is indicated explicitly by the client) system resources are released.

B.The service implementation

We identify three main concerns addressed by the service: QoS negotiation, QoS mapping, and concrete resource manipulation. These concerns are addressed by *generic components* and *plug-in components[Fig 2]*. The generic components provide generic functionality to manage the plug-in components.

Generic Components. Three generic components are exposed to the application layer:

- *QoSRepository* – offers an interface available to clients for registering required QoS with CORBA object references;
- *QOA* (QoS-aware Object Adaptor) – offers an interface available to servers for registering offered QoS on a CORBA object and its servant;
- *GenericNegotiator* – encapsulates the general protocol for performing client-initiated explicit negotiation. The negotiation is performed by the client using an object reference that has been registered with the QoSRepository before.

Two generic components are responsible for managing plug-ins, the *MapperManager* and the *ResourceManager*. The *GenericController* is responsible for managing the control mechanisms for sustaining negotiated QoS sessions. None of these components are exposed to the application layer.

The *MapperManager* interface offers installation, removal and reference retrieving of QoS Mappers. Each QoSMapper registered with the *MapperManager* is identified by a unique identifier.

The *Resource Manager* is responsible for managing various resources within the middleware as well as from the software layers below the middleware (typically, operating systems and transport protocols). Thus, a resource may represent a number of dispatcher threads, but also a complex relation between network delay and network throughput.

The *GenericController* is responsible for managing the specific controllers in each plug-in, so that once a contract has been negotiated, QoS can be successfully sustained. The generic controller is deployed only at the client side of the service. The application layer can only access the control mechanism through a callback registered by the client during the establishment of required QoS parameters. This callback is used by the service to notify the application about the changes in the status of a negotiated contract.

Plug-ins. The service allows developers to provide custom QoS support by providing software plug-ins. A plug-in encapsulates mechanisms for mapping one or more QoS dimensions of the user-

level QoS specification onto underlying technological programming APIs.

A plug-in comprises four parts:

- A *SpecificNegotiator* that provides custom negotiation algorithms;
- A *QoSMapper* that encapsulates the mapping of QoS dimensions onto programming APIs;
- One or more *ResourceWrapper*s that encapsulate concrete resource implementations;

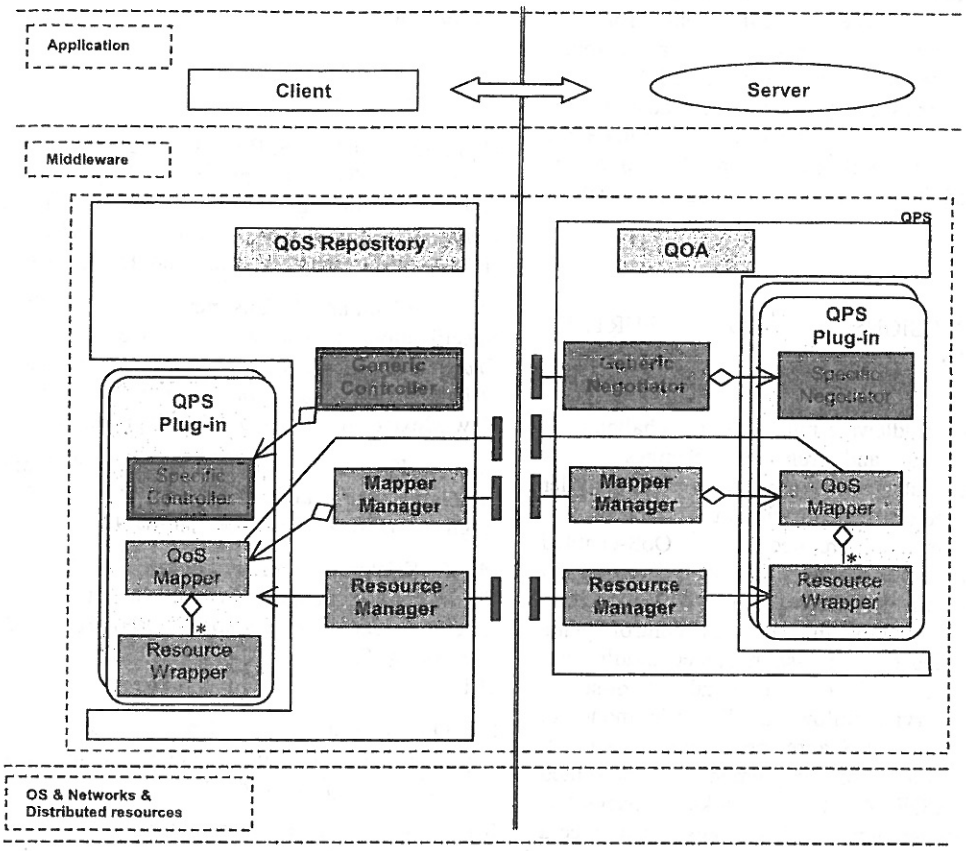

Fig. 2. The client and server sides of the service

- A *SpecificController* that provides mechanisms to sustain the negotiated QoS for a particular plug-in.

The SpecificNegotiator allow developers to supply various alternative strategies for QoS negotiation. The elements of a custom negotiation algorithm are:

- Validation of required and offered QoS specifications.;
- Matching of the required and offered QoS specifications, resulting in an agreed QoS.

A QoSMapper provides algorithms for interpreting application level QoS specification in terms of a set of concrete resource allocations. The developer is free to choose which concrete resource allocations to implement and how to associate them with a QoS specification.

A ResourceWrapper encapsulates programming code necessary for managing one or more resources (e.g., network delay, CPU time, memory). A ResourceWrapper corresponds to one or more QoS dimensions from the QoS specification. The relation between these dimensions is such that one of them cannot be considered separately from the other dimensions. For example, in the RSVP protocol, network delay and data throughput cannot be provided separately because according to the RSVP data structures for establishing of a network reservation these dimensions depend on each other[3].

A SpecificController encapsulates the algorithms necessary for sustaining QoS of a negotiated QoS contract over the time a client wants to use the service. The SpecificController contains sensors that measure values related to the QoS parameters, has a detection mechanism to determine when a contract is violated, and has at its disposal means for adapting the current resource allocations so that QoS can be sustained.

Each plug-in provides concrete resource implementations (ResourceWrappers) to the resource management, and specific negotiation algorithms (SpecificNegotiator) to the GenericNegotiator. The *GenericNegotiator* aggregates specific negotiators from all plug-ins to complete the negotiation algorithm.

C. Using the service

To its users, the service provides two interfaces: the QoSRepository and the QOA. QoSRepository is the interface at the client side allowing the user to set required QoS on an object reference that has been offered from a server object. The QOA is the interface at the server side that allows activation of a CORBA object with an offered QoS specification on its operations. Subsequent object references to this QoS-enabled object allow the use of these references to establish a client/server binding supporting QoS. To developers, the service offers the "QoS services" set of interfaces that enable the management of pluggable QoS dimensions support.

After setting a required QoS on a QoS-enabled object reference, users have to explicitly invoke the negotiation on the QoSRepository. If the service can find a match between the required QoS parameters, the QoS capabilities of the server object, and the available resources of the middleware, the negotiation is considered successful.

During a negotiated QoS session that sustains an agreed QoS, the client may receive notifications from the detection and control mechanisms via a callback, indicating a possible violation of the agreed QoS. The control mechanism tries some strategies for adapting the underlying resources before it decides to terminate the session (the latter happens if resources reach a condition, in which the agreed QoS cannot be supported anymore).

V.CONCLUSIONS AND FURTHER DEVELOPMENTS

Next generation middleware must meet the challenge of evolutionary changes and run-time changes in a heterogeneous distributed computing environment, in order to provide distributed objects with support for QoS.

This paper presents an architecture for QoS-enabled middleware that separates the QoS support functions from 'traditional' data transfer functions. The QoS Adaptive Service is our framework that enables control plane functions to be added to off-the-shelf object middleware, for controlling the QoS of individual client-server associations. The service follows a lifecycle model to establish and control a QoS agreement between a client and a server.The framework implementation presented here uses standard CORBA extension hooks (Orbacus 4.1. a free ORB implementation) , which makes the service a portable CORBA service. Future work includes the study of the applicability of the provisioning service to manage the QoS of multimedia streams and implementing a more advanced interface between the service and system level QoS control mechanisms, including other QoS networking mechanisms (as Diffserv protocol).

Next generation middleware must meet the challenge of evolutionary changes and run-time changes in a heterogeneous distributed computing environment, in order to provide distributed objects with support for QoS. This paper presents an architecture for QoS-enabled middleware that separates the QoS support functions from 'traditional' data transfer functions. The QoS Adaptive Service is our framework that enables control plane functions to be added to off-the-shelf object middleware, for controlling the QoS of individual client-server associations.

The service follows a lifecycle model to establish and control a QoS agreement between a client and a server.The framework implementation presented here uses standard CORBA extension hooks, which makes the service a portable CORBA service. Future work includes the study of the applicability of the provisioning service to manage the QoS of multimedia streams and implementing a more advanced interface between the service and system level QoS control mechanisms, including other QoS networking mechanisms.

VI. REFERENCES

[1] F. Fitzpatrick, G.S. Blair, G. Coulson, N. Davies and P. Robi*n* (1998) "Supporting Adaptive Multimedia Applications through Open Bindings", *4 th International Conference on Configurable Distributed Systems* (ICCDS'98), Annapolis, Maryland, USA, pp56-64

[2] S. Frolund and J. Koistinen (1999) "Quality of Service Specification in Distributed Object Systems Design", *Proceedings of the 4 th USENIX Conference on Object-Oriented Technologies and Systems (COOTS),* Santa Fe, New Mexico, April 27-30, 1998. Applications (DOA'99)

[3] M. Karsten, J. Schmitt and R. Steinmetz (2001) "Implementation and Evaluation of the KOM RSVP Engine", *IEEE InfoCom 2001,* pp90-98

[4] T. Kristensen and T. Plagema*n* (2000) "Enabling Flexible QoS Support in the Object Request Broker COOL", *20th International Conference on Distributed Computing Systems (ICDCS'00),* Taipei, Taiwan, pp96-101

[5] Object Management Group (2000) The Common Object Request Broker: Architecture and Specification *OMG document formal/00-10-33.*

[6] Object Management Group (1999) Portable Interceptors, *OMG Document orbos/99-12-02*

[7] T. Plagemann, F. Eliassen, B. Hafskjold, T. Kristensen (2000) "Flexible and Extensible QoS Management for Adaptive Middleware". *International Workshop on Protocols for Multimedia Systems(PROMS 2000),* Cracow, Poland

[8] D.C. Schmidt (1997) "Acceptor and Connector: Design Patterns for Initializing Communication Services", in *Pattern Languages of Program Design (R. Martin, F. Buschmann, and D. Riehle, eds.),* Reading, MA, Addison-Wesley, pp 345

[9] Siqueira and V. Cahill (2000) " A QoS Architecture for Open Systems" , *20 th International Conference on Distributed Computing Systems (ICDCS'00),* Taipei, Taiwan, pp78-84

[10] J. Zinky, R. Schantz, J. Loyall, K. Anderson and J. Megquier (2001) " The Quality Objects (QuO) Middleware Framework" . *Workshop on Reflective Middleware (RM 2001),* New York, USA