

Middleware Adaptive Management and Control for End-to-End QoS

Alia K. Atlas, Joseph P. Loyall, and Richard E. Schantz

{akatlas, jloyall, schantz}@bbn.com

BBN Technologies

<http://www.dist-systems.bbn.com/tech/QuO>

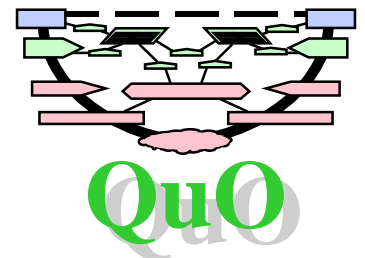
Outline

Motivation & Context

Overview of QuO technology

Desired Features of Network API

Conclusions and Issues



As real-time applications become more distributed and complex, their needs for knowledge and control increase.

- Real-time applications always require resource management and allocation.
- Traditionally, resources (CPU, memory, I/O) are controlled at run-time but allocated at design time.
- More complex applications and environments require adaptive behavior to deal with uncertainty.
- Distributed RT applications must deal with an unpredictable network, as well as control environment.

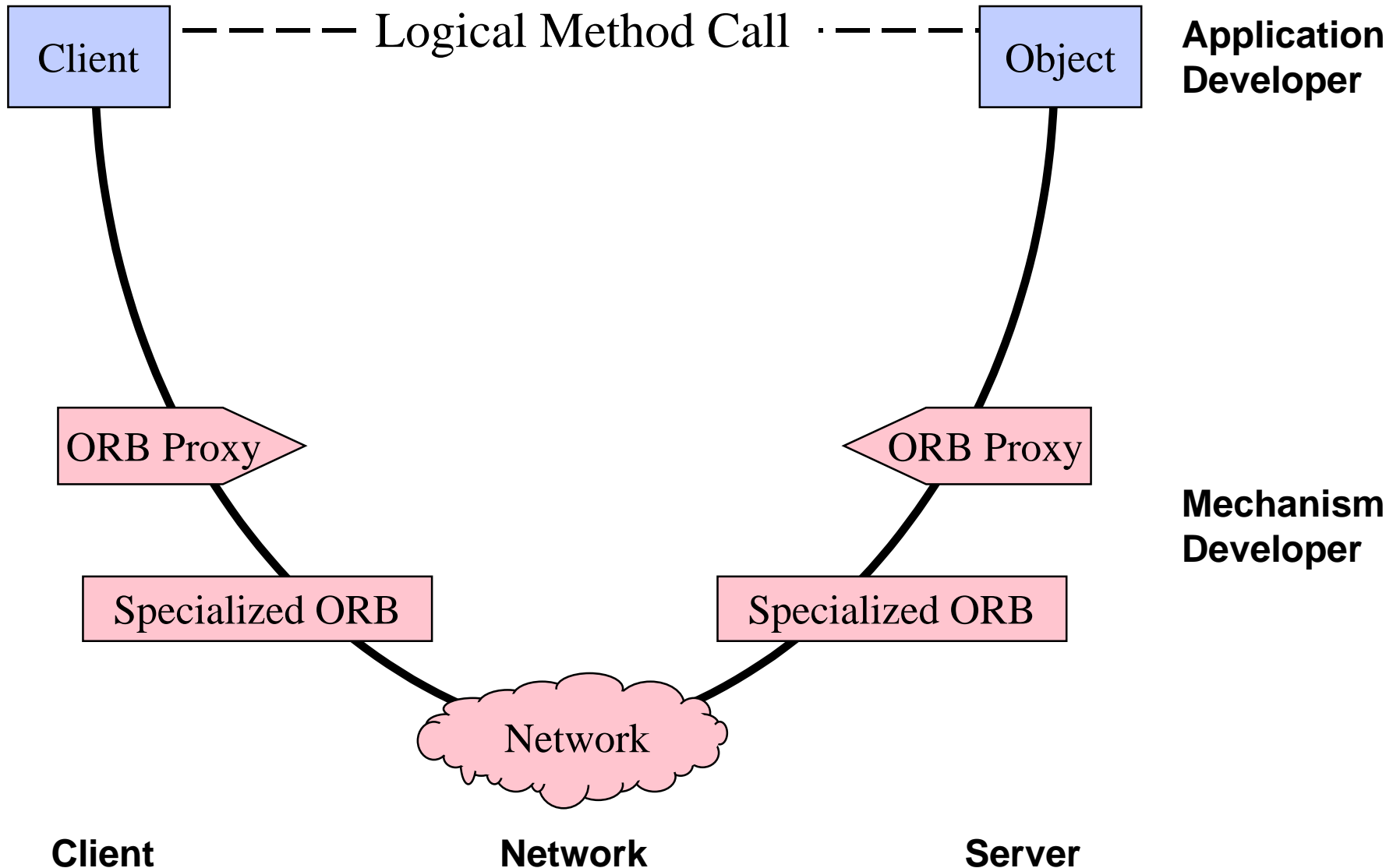
Resource Allocation and Control are Common to RT Applications - Solve Problem Once with Middleware.

- Adaptive resource allocation must be supported by all applications in a system.
- Resource management requires low-level control, which ties applications to a specific platform.
- Adaptive applications must support operation with variable available resources.
- Applications should monitor delivered QoS to determine current resource requirements.

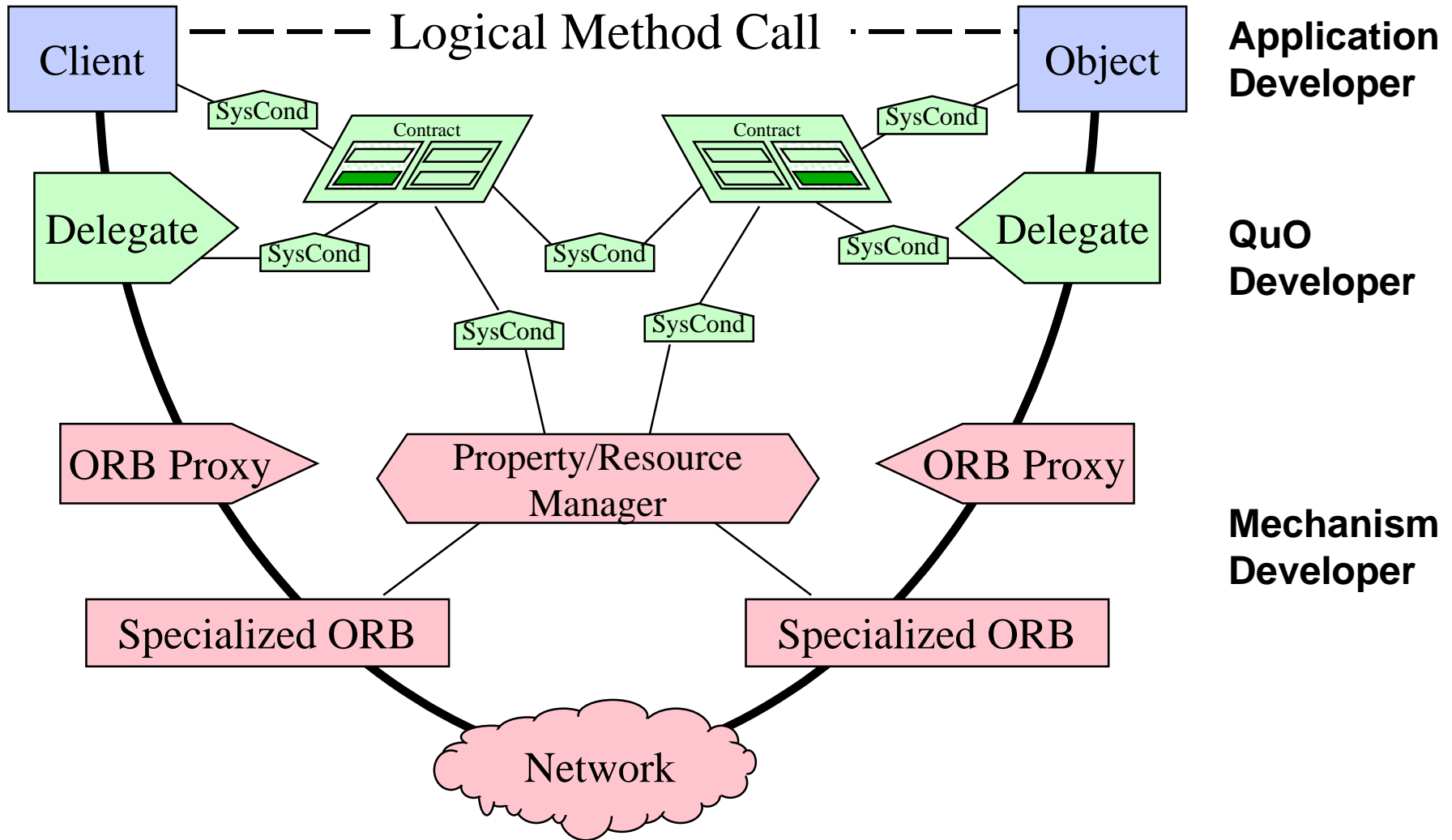
We are building upon research that we're conducting in adaptive distributed systems

- Developing technology to support specification, control, and measurement of QoS in distributed systems, including real-time, dependability, performance, and resource management
- Addressing the following problems of critical, networked applications:
 - No effective way to control behavior of critical applications in today's highly networked environment
 - Gap between low-level mechanisms that control resources and high-level strategies appropriate for critical applications
 - Researchers generally working on one piece of the solution, e.g.,
 - Focus on one critical QoS property: real time behavior, security, fault tolerance, ...
 - Focus on implications of one time epoch
 - Functional integration has taken precedence over system properties
- End goal is to support building integrated QoS adaptive applications (with varying requirements on granularity of changing behavior):
 - adaptable: change at runtime while application/service running
 - reconfigurable: change at runtime while application/service halted
 - evolvable: change at development time

Simplified DOC (CORBA) Runtime Components



QuO adds specification, measurement, and adaptation into the distributed object model



A QuO application contains additional components (from traditional DOC applications)

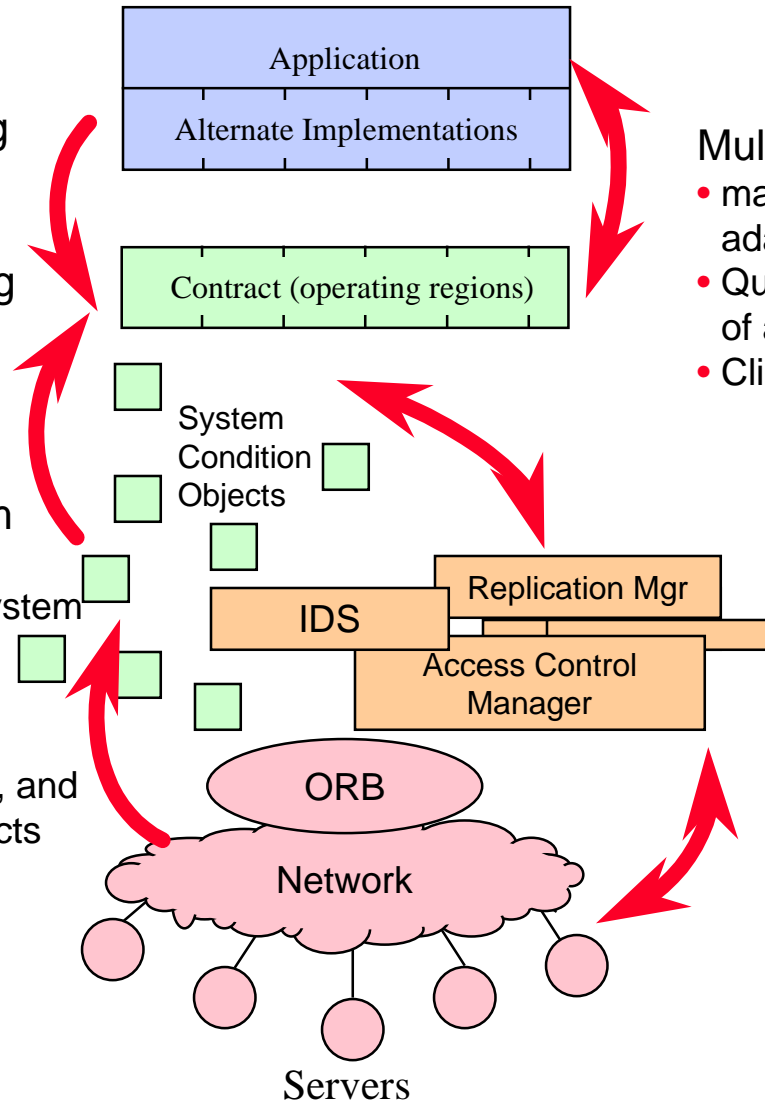
- **Contracts** summarize the possible states of QoS in the system and behavior to trigger when QoS changes
 - Regions can be nested, representing different epochs at which QoS information becomes available, e.g., *negotiated regions* represent the levels of service a client expects to receive and a server expects to provide, while *reality regions* represent observed levels of service
 - Regions are defined by *predicates* over system condition objects
 - *Transitions* specify behavior to trigger when the active regions change
- **System condition objects** are used to measure and control QoS
 - Provide interfaces to system resources, client and object expectations, mechanisms, managers, and specialized ORB functions
 - Changes in system condition objects observed by contracts can cause region transitions
 - Methods on system condition objects can be used to access QoS controls provided by resources, mechanisms, managers, and ORBs
- **Delegates** implement the QoS specific adaptivity behavior
 - Upon method call/return, delegate can check the current contract state and choose behavior based upon the current state of QoS
 - For example, delegate can choose between alternate methods, alternate remote object bindings, perform local processing of data, or simply pass the method call or return through

QuO applications specify, control, monitor, and adapt to QoS in the system

Specification of operating regions, alternate implementations, and adaptation strategies using QuO's QDL

System condition objects **monitor** QoS in the system

- system condition objects recognize changes in the system and notify the contracts that observe them
- QuO contracts notify client programs, users, managers, and other system condition objects through transition behavior



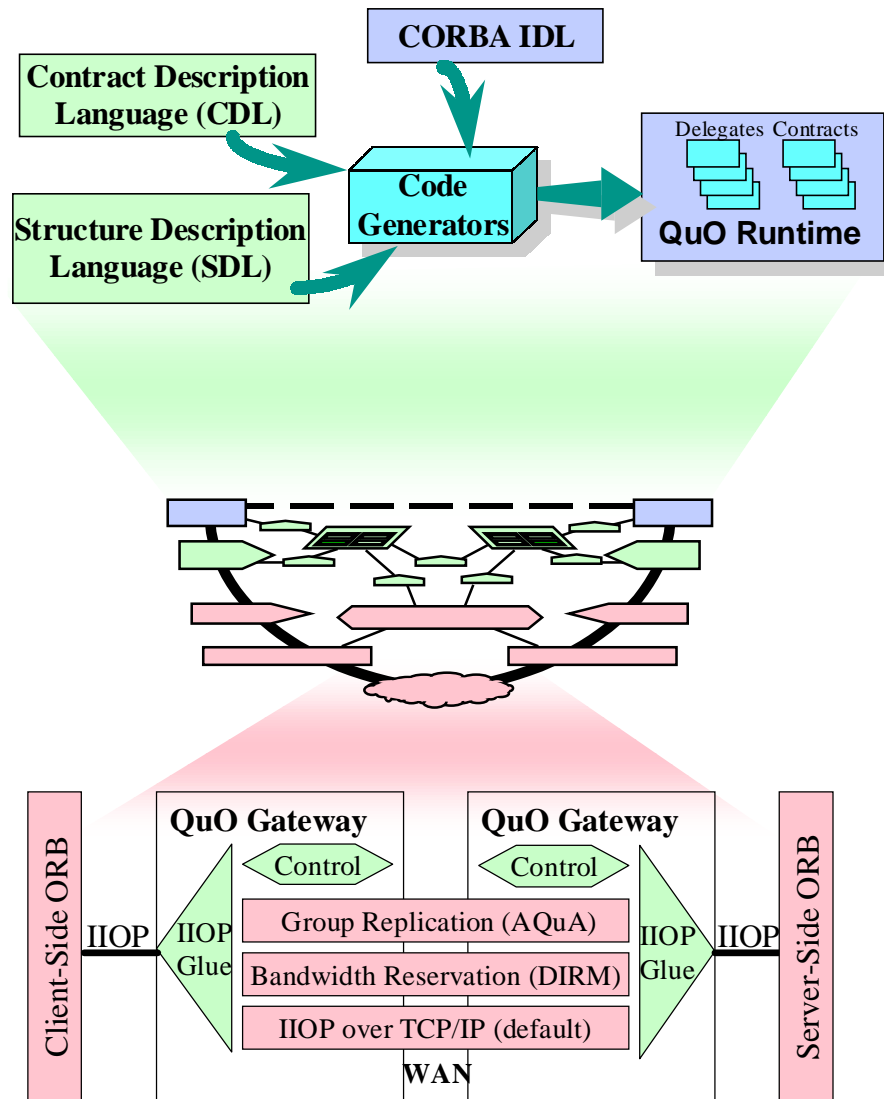
Multiple layers of **adaptation**

- managers and mechanisms can adapt to changes in the system
- QuO contracts provide another layer of adaptation
- Client and user can also adapt

Mechanisms and managers **control** QoS in the system

- a layer below QuO that provides ORB-level services, such as managed communication, replication, or security
- contracts and delegates interface to these services through system condition objects

The QuO Toolkit provides tools for building adaptive applications



- Quality Description Languages (QDL)
 - Contract description language, adaptive behavior description language, connector setup language
 - Code generators that generate Java and C++ code for contracts, delegates, creation, and initialization
- System Condition Objects, implemented as CORBA objects
- QuO Runtime Kernel
 - Contract evaluator
 - Factory object which instantiates contract and system condition objects
- Instrumentation library
- QuO gateway
 - Insertion of special purpose transport layers and adaptation below the ORB

Benefits of QuO adaptable software to RT distributed applications

- QuO framework separates functional aspects from QoS aspects
- Contracts and System Conditions support adaptivity
- Provides interface to Resource Managers
 - Applications can negotiate for the desired resource
 - Resource managers can notify applications of decisions and adapting resource conditions
- Applications can run in a variety of environments with different resource availabilities

Resource Manager Requires Knowledge of Network

- Awareness of quality of the current network
 - Low bandwidth could trigger transition to a CPU intensive mode
- Knowledge of network options
 - Critical traffic could take low latency, low bandwidth path, while regular traffic takes higher latency, higher bandwidth path

Middleware can trigger adaption to network conditions —
but only if they are known.

Specification of Traffic Differentiation

- Resource Managers determine an application's share of the managed resources.
- Different applications need to use network differently
- Need semantics to express both traffic importance and desired QoS attributes.
 - Critical vs. non-critical
 - Low latency
 - Expected bandwidth

Middleware Can Express Desired QoS and Resources to Network

Resource Managers Must Know Network Costs

- If there is no advantage to not using all of an available resource, then all of it will be distributed.
- Network cost creates another dimension which must be managed by a resource manager.
- Trade-off benefit of average higher resource utilization versus lower average with reserve for critical periods.

Conclusions

- We have developed middleware that supports adaptable software that can measure and control changing system properties
 - Supports software that has critical resource needs and can be deployed in unpredictable WAN environments
 - Instrumentation, triggers and effects can be defined and employed in an application in a structured manner as opposed to ad-hoc exception handling
 - Enables the combination and trade-off of multiple QoS dimensions, e.g., real-time, dependability and security
- This is useful for distributed RT applications
 - Supports applications that can adapt to changing external environments
 - Supports application-level interfacing of resource managers

Now, the middleware needs the network to provide interfaces, information and control.

Where to find more information

- QuO

<http://www.dist-systems.bbn.com/tech/QuO>

- To get the QuO Toolkit v2.1 software, send e-mail to quo-help@bbn.com