

# The Quality Objects (QuO) Middleware Framework

A Position Paper for QoS in Distributed Object Systems Workshop (ECOOP 2000)

June 13, 2000

**John Zinky**

Jzinky@bbn.com

BBN Technologies' Quality Objects (QuO) research is extending the distributed object middleware paradigm to enable the development of adaptive distributed applications. QuO supports the development of applications that can perform the following:

*Specify the level of service* they desire. That is, in addition to specifying functional interfaces, an application can specify its systemic (non-functional) requirements, such as real-time response, memory utilization, access control, synchronization, managed bandwidth, or dependability.

*Specify behavior alternatives* and strategies for adapting to changing levels of service. An application can choose different behaviors based upon its current operating mode, the level of service being provided by the system, etc. These adaptive behaviors can be used to recover from problems, to proceed in the face of degraded service, to implement different processing modes, and to enhance portability among different configurations.

*Measure and control* system resources, conditions, and mechanisms. QuO provides feedback to the application about the state of the system and standard interfaces (called system condition objects) to resources and mechanisms. The application can use the feedback information to trigger adaptation or reconfiguration. Likewise, the application can use the control interfaces to try to achieve the desired service.

These capabilities go beyond those provided by conventional distributed object middleware. The QuO framework consists of the following components:

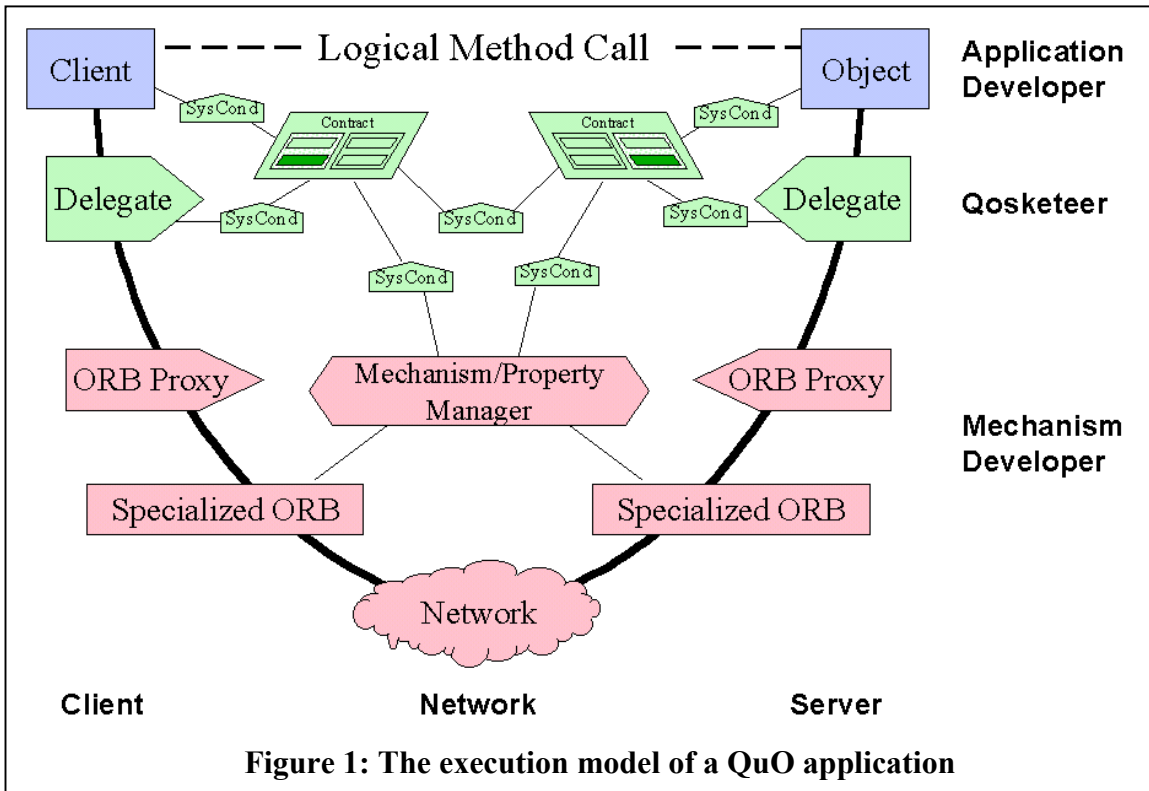
*Description languages* for describing contracts between clients and objects in a distributed system, adaptive behavior alternatives and strategies, and probes into the system to measure and control QoS mechanisms and resources.

*Code generators* that create and distribute executable (currently Java and C++) code throughout the application and system.

*A runtime kernel* that organizes, schedules, and manages contract evaluation, feedback, and updates.

*Reusable libraries* of system condition objects that interface to system resources, mechanisms, and managers.

In a traditional distributed object application using the CORBA model, a client makes a method call on a remote object through its functional interface. The arguments to the call are marshaled by an ORB on the clients host, delivered using a standard protocol (such as CORBA's IIOP) to an ORB on the objects host. The remote ORB unmarshals the data and delivers it to the remote object, which executes the method. The method's out parameters and return value, if any, are marshaled by the remote ORB, delivered to the client's local ORB via the standard protocol, unmarshaled by the local ORB, and delivered to the client.



As illustrated in Figure 1, a QuO application also includes the following:

*QuO contracts* summarize an application's current operating mode, expected resource utilization, rules for transition among operating states, and means for notifying application of changes in QoS or in system status. Contract specifications are written in a high-level specification language (not raw Java or C++ code) and preprocessed into compilable code by code generators.

*System condition objects* provide interfaces to system resources, mechanisms, and managers. They provide standard, high-level, and reusable interfaces to measure, manipulate, and control lower-level real-time control and measurement capabilities. They export values that describe facets of system status, such as the current memory utilization or priority of the running thread, and provide interfaces to control system characteristics, such as modifying the processor clock rate or scheduling priorities.

*QoS-aware delegates* are adaptive components that modify the systems runtime behavior along the paths of method calls and returns. These help QuO support code adaptation triggered by contract region transitions or by instrumentation packages for measuring performance.

More information about QuO, including copies of published papers and QuO V2.1 software, can be found at: <http://www.dist-systems.bbn.com/tech/QuO>.