28

Managing Systemic Meta-Data for Creating QoS-Adaptive CORBA Applications*

John A. Zinky and David E. Bakken
BBN Systems and Technologies, {jzinky,dbakken}@bbn.com
10 Moulton St. Cambridge, MA 02138 USA

Abstract

Systemic meta-data is crucial for creating adaptive distributed applications. But managing systemic meta-data is extremely complex and currently is done either *ad hoc* or simply ignored. QoS savvy CORBA middleware is the appropriate place to manage systemic meta-data. In this position paper we outline the issues involved in this management.

Keywords:

Quality of Service, QoS, meta-data, CORBA, Quality Objects, QuO.

1. Introduction

Wide area and mobile environments exhibit a large variation in available resources and delivered quality of service (QoS). Distributed applications must therefore adapt to changes in their environment if they are to be effective under such extreme conditions. To adapt, applications must change their behavior at runtime based on the state of its environment. A basic requirement for such adaptivity is access to information about the underlying resources and application requirements. This *systemic meta-data* describes how applications use resources, their QoS requirements, the status of resources, and allocation policies. Without this information applications are rigid because they can only run in a limited and relatively predictable environment.

Obtaining this systemic meta-data requires knowledge of *how* the system is implemented and deployed. Unfortunately, the traditional way in which distributed applications are developed is by using a black box abstraction which specifies *what* the component does — its functional interface — but hides *how* it was implemented. For example, CORBA's Interface Description Language (IDL) defines a functional interface for a remote object; e.g., a method to sort an array with a given argument signature. This black box approach works when *a priori* assumptions of the environmental conditions matches reality and when these conditions do not change. For a large class of environments the black box approach works fine, such as with programs contained in one process, and even in distributed programs working over local area networks (LANs). But, experience has shown that the black box approach is not adequate for all systems, especially distributed system running over a resource constrained network.

* This research was partially funded by DARPA ITO under Contracts F30602-96-C-0315 (AQuA Project) and N66001-96-C-8529 (DIRM Project).

Building QoS into Distributed Systems A. Campbell & K. Nahrstedt (Eds.)

Abstractions of resources themselves hide their true distributed nature in order to provide an idealized in-core functional interface. For example, the socket interface provides a communications abstraction which allowed applications and networks to evolve independently (Fig. 1).

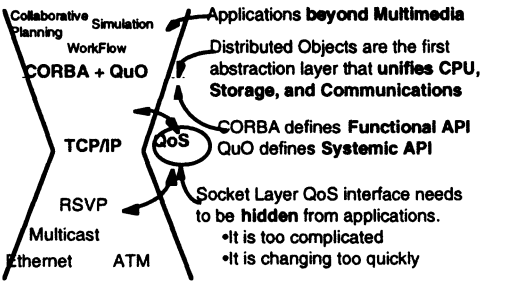

Figure 1: Distributed objects with QoS extensions is a powerful abstraction layer on which to build adaptive distributed applications.

This black box interface for application components and resources must be "opened up" to expose their internal structure because the functional interface is not enough to create adaptive distributed systems. Open Implementation research efforts focus on this problem for a single program [1]. In this approach, an additional systemic interface is used to map an application onto its environment in a workable fashion.

In distributed systems many aspects of the internal development and fielding of an application are more explicit. They include the system properties, the roles of those involved (clients, objects, end users, programmers, etc.), commitment times (runtime, compile time, etc.), as well as the mechanisms to move this information around (available from a server, measured automatically, configuration parameter, etc.). The current state of development tools to support this is unwieldy and unintegrated and is a thus a great impediment to development and deployment of realistic applications over the wide area.

2. Possible Solutions

One solution is to try to provide in-core guarantees for distributed interactions; e.g., making a remote object invocation behave like a local procedure call. However, we believe this is impossible to provide, because it will be deficient in one or more of the following: high latency, prohibitive cost, or unworkable complexity. Pursuing this holy grail has driven much research in the last two decades, but it has become obvious that it is unachievable.

A second solution which is much more achievable and features a much bigger payoff is to make the application more tolerant of the intrinsic limited guarantees (in the spirit of ISO QoS Compulsory levels of agreement [4; Sec. 7.3.2.4]) instead of trying to make the distributed infrastructure meet unobtainable system properties of a single host. This can be accomplished mainly by recognizing loose requirements; providing mechanisms with different, explicit resource tradeoffs; and by knowing when throwing more resources at the problem will help.

Our Quality Objects (QuO) framework adopts this second approach by using open implementation techniques in the context of CORBA [2,3]. QuO provides a clean separation of functional and system issues. Elements of QuO are: contracts to define the interaction between clients and objects, system condition objects to observe the environment, and layers of delegates to mask variation.

3. Structure of Systemic Meta-Data

The basic life cycle of systemic meta-data is: first collecting, reducing, and summarizing it; disseminating it to interested parties; analyzing it; and acting upon it. However, providing this life cycle in a distributed system involves many facets, including:

Mechanism: Systemic meta-data can travel between locations using several mechanisms supported by the underlying communications infrastructure (Figure 2).

Commitment epochs: As time progresses, a distributed system accumulates knowledge about its environment. E.g., at design time the different implementations are known, while at runtime the availability of a resource can be ascertained.

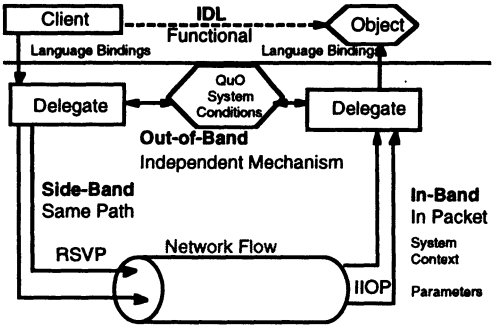

Figure 2: Meta-data about system properties are transferred via disparate mechanisms

Roles: Distributed systems are created, deployed, and used by multiple personnel, often in different organizations, with each responsible for a subset of the application.

Location: Distributed systems *ipso facto* have multiple locations with high noise, limited bandwidth, and long latency between them.

Granularity/Accuracy: Fundamental tradeoffs exist between obtaining accurate systemic meta-data and the resources and effort needed to obtain a level of accuracy.

Kind: Multiple properties (ISO QoS characteristics [4]) such as availability, security, real-time, resource allocation need to be provided in an integrated fashion.

Managing the interrelationship between all these facets is very complicated and is well beyond the capability of application programmers. We thus need QoS middleware which provides a framework to make these facets explicit and to help manage them.

4. QuO Support for Adaptive Applications

QuO is a framework which makes these facets explicit with the following components (Fig. 3):

System Condition Object Explicit objects which capture and maintain a specific piece of systemic meta-data. QuO supports a spectrum of system condition objects, from objects local to the client to remote CORBA objects. For example, one system condition object could measure the client's method invocation rate while another may represent an interface to an RSVP session.

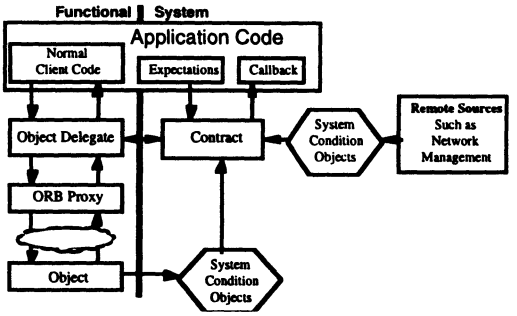

Figure 3: QuO integrates systemic meta-data through the use of system condition objects, contracts, and delegates.

Contracts: QuO contracts summarize system condition information and organizes them based on commitment epochs. For example, at negotiation time a contract may expect a client's throughput to match network capacity. At runtime, however, these may diverge, and QuO provides hooks for recognizing this.

Delegates: Delegates change the behavior of the object based on the region of the contract. For example, if the client invocation rate is greater than the capacity the delegate may block an invocation to implement traffic shaping.

5. Areas for Future Research

Supporting systemic meta-data falls under several areas of research. Networks need to support explicit mechanisms for moving and storing systemic meta-data. Application programmers need APIs for QoS at the client/object boundary rather than at the socket level. Finally, the semantics of the systemic meta-data itself needs to be codified so in the future we can automatically synthesize and reason about QoS.

References

[1] Kiczales, Gregor. Beyond the Black Box: Open Implementation, *IEEE Software*, January 1996.

[2] Zinky, J., Bakken, D. and Schantz, R. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, Special Issue on the OMG and CORBA, 3(1), April, 1997.

[3] QuO, *Internet Publication*,URL `http://www.dist-systems.bbn.com/tech/QuO`

[4] ISO/IEC 13236 – Quality of Service – Framework – DIS July 1996