

# Middleware Design Issues for Application Management in Heterogeneous Networks

Y. Tian, S. Frank, V. Tsaoussidis, H. Badr

Department of Computer Science, SUNY at Stony Brook, NY 11794-4400

## Abstract

*With the increase of the heterogeneity of Internet and the complexity of distributed computing, application management is facing more and more sophisticated requirements. In a dynamic, synergistic, and geographically broad environment, cross-domain resource management is becoming an important factor for application Quality of Service.*

*We present a middleware architecture and initial implementation of that middleware that satisfies applications according to expected requirements, Application Service Level Agreement and distributed networked resources in and beyond the local environment. The networked resources are grouped in a hierarchical manner and their usage is optimized through the resource management system. This framework presents the same interface for any given resource type, with only the parameters changing to reflect that resources' specific requirements. This system enables interoperability, dynamic service discovery, and management functions such as monitoring, controlling and reporting, and dynamic reconfiguration and management.*

## 1. Introduction

Computer networks by nature are dynamic environments where many users and applications share resources. Application management in networked environments is associated with effective management of networked resources with respect to their static characteristics as well as the dynamic changes of their state. During the past years, management systems have tried to overcome the heterogeneity of network resources, operating systems, and protocol of different domains, resulting in standard protocols like SNMP that are widely used today. When later application management become an important issue, SNMP-MIB was extended to include application-related resources. The demand for Quality of Service (QoS) raised by multimedia applications, introduced resource reservation protocols like RSVP as additional standards.

We have implemented an application-oriented resource management system using Common Object Request Broker Architecture (CORBA) [16]. Resource management has common functionality that is independent of resource type. CORBA [10, 11] is intended to support the production of

flexible and reusable distributed services and applications. Our framework supports management operations to guarantee QoS for a variety of resource types, while possessing a uniform operation interface at the application level. This system allows a plug-in approach for various resources.

A similar goal is also a major research topic of active and programmable networks [15]. This architecture enables users to inject customized programs into the nodes of the network in which the switches perform customized computations on the messages flowing through them. The development of active networks is motivated by the increasing requirement of user-driven computation and by the emergence of mobile code technologies that make dynamic network service innovation available. Dynamic resource management and adaptive QoS is another related field, which focuses on developing resource management mechanisms with different kinds of customized services. Some recent approaches [4, 5, 6, 13] are based on dynamic management using CORBA. Some other approaches are based on management rules for specific protocols of the network or of other layers [8, 9]. An SNMP-based network management schema using CORBA was proposed in Bell Laboratory [8]. The significant results of these studies can be combined with the system presented here.

BBN is developing the Dynamic Integrated Resource Management (DIRM) project [5] focusing on the network management and control of resources from the Application Programmer's Interface (API) perspective. The API allows applications to control QoS for their communications over RSVP. The results of this research are expected to assist applications to integrate resource reservations, as well as multicast capabilities. CORBA interfaces are used to overcome the heterogeneity of system specific control mechanisms.

A recent work at NEC C&C Research outlines the QoS requirements from the user's perspective [12] and suggests a framework to satisfy user requirement [13]. This work is moving towards the direction of application-oriented QoS, and its focus is on the mechanism and framework which enable QoS benefits to be achieved from different layers of the network architecture appropriately managing the network resources. An application-oriented framework for resource management, and appropriate management interfaces to reflect QoS parameters, are presented in [17, 18].

The Darwin project [1, 2, 3] developed in Carnegie Mellon University aims at the design and implementation of a set of resource management mechanisms that support the deployment of customizable, value-added services in the network by using Service Brokers. Once a broker has identified the resources needed to satisfy a request, it uses a signaling protocol to allocate these resources. This work is very interesting; however, it differs significantly from our work in that its proposed framework is more an allocation service than a management system. The focus there is on the functionality of the “service brokers”.

In our system, the focus is on the uniform operational interfaces, the management operations, the widespread optimal use of resources according to a best effort service, and load balancing beyond the borders of a LAN. We consider the impact of this system on the network traffic as well. For testing purposes, the simulation of resources is made on the behavior of a switch, even though the resource can be any general type of resources such as printers, cameras, or software resources such as application specific protocols, etc.

In section two we present the functionality of the system, in which we first describe in detail the session types available and then explain the resource information scheme. In section three we describe the architecture of the system, starting with a description of a typical scenario in order to illustrate the timing and communication within and across domains. In section four we detail the implementation of a case example where we present implementation details for switch management. In section five we present some concluding remarks.

## 2. Functionality

We implemented the proposed system using the TAO ORB [14]. Object-oriented techniques offer a variety of principles, methods and tools that help to alleviate much of the complexity associated with developing distributed applications. For our prototype we use the Event, Naming and Trading Services.

In our framework, the whole network-computing environment is constructed as a hierarchical resource tree. Depending on the size of the network-computing environment, this resource tree can have two or more levels. At the leaves of the resource tree, the resources are grouped according to our standard grouping criterion and the resource location. Each Resource is managed by a local entity called Resource Inspector. The tasks of Resource Inspector include monitoring/controlling the status of the resource, providing a uniform interface to the higher level satisfying the application request.

### 2.1. Resource requirement sessions

Three different resource requirement sessions can be raised from the user application:

#### 1) *Guaranteed QoS*

The requirement is specified in terms of parameters of services. For example, for a switch-like resource, these parameters would be bandwidth, delay, reliability and jitter. The return code for this application requirement is simple: it informs the user application whether the request was completed successfully, appended to a waiting list or rejected.

#### 2) *Priority level QoS session*

The user desires a certain level of service from the system, but with no particular parameters as the case in the first session. The return code is similar to the first session, with the addition of the Resource Inspector's object reference.

#### 3) *“Best effort” service*

The user wishes to have the best possible results from the system as a whole. The service request message is tagged with the current values of the “benefit” function and the signature of the Resource Inspector that can provide it. Replies are of the same format as for the priority level QoS sessions.

## 2.2. Resource information scheme

An important characteristic of the Resource Inspectors is that, although their management mechanism is resource and/or system specific, their operational interface as well as the reporting mechanism is uniform and interoperable at the application level. A node can be defined for each domain, such as a building in a University campus. For each node, there is a Node Inspector, which is in charge of the admission control, resource allocation, failure recovery and other management functionality for this domain. It is quite often the case that resources in a local group can't satisfy certain requests. In such cases, it is the Node Inspector's task to consult with other groups of resources until the request can be satisfied.

Two variants of the resource information scheme were considered. One was a centralized information strategy, where the Node Inspector keeps current status on each resource. The other is a decentralized strategy, where the Node Inspector has only *available* and *busy* status of the resources. The first approach has the potential advantage of quick decision making upon a resource request. The Node Inspector keeps detailed information, such as the bandwidth, delay, reliability and jitter for a switch. However this design has a major drawback in the network traffic overhead it will produce. Since states might change rapidly for many resources, updating the system with detailed information could potentially add a great deal of traffic. Moreover, if a state change occurs at the resource level during the request processing, the information may not be current anyway. The preferred approach minimizes the bandwidth usage by the resource management framework. The Node Inspector is notified when a resource becomes available and subsequently if it becomes busy (as defined below), but it is not notified of the exact current state of the resource. The resource allocation decision is delayed until the Node Inspector queries

the appropriate Resource Inspectors. This communication schema causes traffic after each application service request is raised, however, in most common cases, such as management of switches, application service requests occur much less frequently than do resource status changes. It is noteworthy that the “rough” information prevents the Node Inspector from checking resources that are clearly not able to meet a QoS agreement. When resources do not meet the basic service criteria, no further updates are sent. This policy reduces significantly the management overhead.

The selection of a resource consists of three steps: selecting all suitable resources, consulting these suitable resources and making the final selection. We define the criterion of *busy* for each resource type in terms of its resource-specific parameters. The Resource Inspector maintains this information, which is local to the resource. Upon a request, only resources of the correct type that are *available* and *not busy* need to be queried. The final resource selection, and hence the Resource Inspector’s response to the query, depends on the QoS type. For Guaranteed service, the resource is chosen from among those that can meet the requirements. For Priority and Best Effort service, the Resource Inspector calculates the benefit function value (weighted sum of different factors) and includes this value in its reply to the Node Inspector. After comparing these benefit function values, the Node Inspector makes the final resource selection. Since the administrator-provided benefit function may differ from one request type to another a static table for all resource types would not work. For example, for a printing job, the benefit function might be a weighted sum of different factors such as speed and print quality; the benefit function for a switch might be a function of total throughput and maximum delay and/or jitter. Therefore, each Resource Inspector corresponds to a certain type of resource (e.g., cameras, printers, scanners, or network bandwidth).

### 3. Architecture and implementation

The implementation and the communication model are shown in Fig.1. Naming, Trading and Event Services are combined to satisfy the model’s functionality requirements. The Naming Service is used as a “glue” between the application, the Node Inspector and Resource Inspector, so they can access each other by “name” instead of by object reference. At the heart of the system lies the Trading Service; it provides a matchmaking service for objects. Resource Inspectors, as service providers, use it to advertise their services. The Trader is the object that performs the Trading Service and the Event Channel is the object that performs the Event Service. The Node Inspectors consult the Trading Service to obtain information about suitable services offered by Resource Inspectors, based on the type of service and specific characteristics of each service, the resource capability, and the user requirements. They then assign the appropriate resource to the user. The Resource Inspectors

export their service offers, while Node Inspectors import the suitable ones:

```
interface Export{offerId export();}
interface Import{void query();}
```

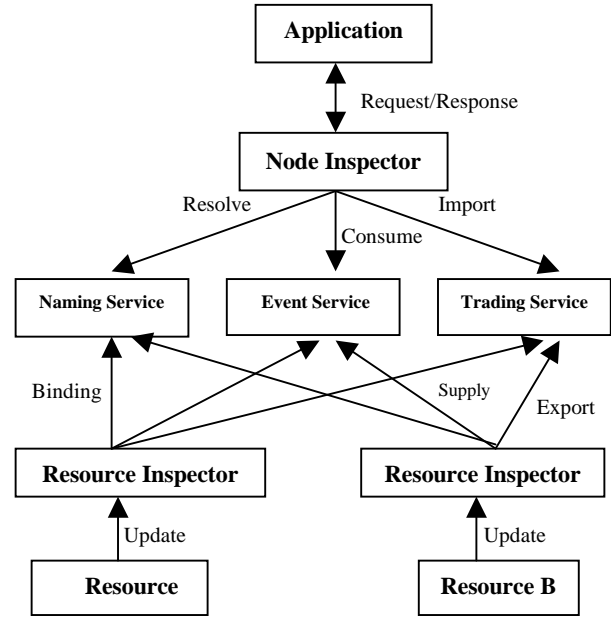


Figure 1: Implementation and communication model

The basic scenario of a best effort request is explained below:

1) During initialization the Node Inspector resolves the Node Inspector name through Naming Service. The Resource Inspector registers its service to the Trader, from which the Node Inspector can import services provided by the resource.

2) As regular actions of the system, the Resource updates its status to the Resource Inspector when the status changes. The Resource Inspector updates its availability and state information to the Node Inspector through Event Service only when the availability of the resource or state information changes (i.e., from *available* to *down* or from *busy* to *not busy*, respectively).

3) When a specific service is requested by an application, the Node Inspector checks the Service Level Agreement and then checks the benefit function of each *available* and *not busy* resource through the Event Channel.

The Resource Inspector registered with the corresponding Event Channel calculates the “benefit” function and reports back to the Node Inspector through Event Channel. The Node Inspector chooses the best Resource Inspector according to the “benefit” functions and grants the Application’s service request. The service data flows to the Resource.

4) If the service request can not be satisfied at the local domain, the Node Inspector consults with other domain’s Node Inspectors for service. The Node Inspector’s recursively

apply to other Node Inspectors until the service request can be met or has been denied by all nodes.

#### 4. A case example

We simulate a common switch as a prototype resource as well as the applications. We present an operation overview in Fig.2. The application simulation generates packets and the request of services from the Node Inspector. The simulated switch accepts the data flow and does status reporting.

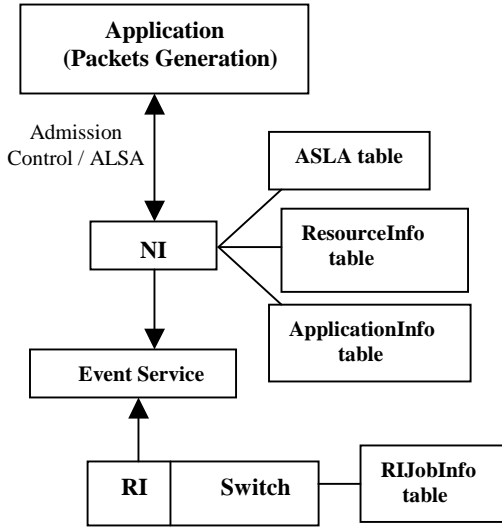


Figure 2: Testing of the system using switch as resource

When a new packet stream is initiated, the Node Inspector is called by the Packet Stream Generator to find a resource with enough available bandwidth, and which can meet the Quality of Service requirements. If one is found, the resource name and a valid JobId are returned. Otherwise, JobId is set to -1 to indicate a failure. The switch uses queues to controls the Quality of Service. A priority level determines the queue for the packets of a particular customer. Lower numbered level queues always take precedence over higher level ones. The Quality of Service parameters of Delay, Jitter and Reliability are only valid for level 0 customers. The switch will trap to the Resource Inspector if the QoS for any level 0 customer isn't being met, or if the bandwidth reserved is exceeded or the bandwidth exceeds the "busy" threshold. The Event Channel is used for this.

In the following paragraphs, we present the specific details of the system components of our case example.

##### 4.1. Node inspector

The Node Inspector is implemented as a standard CORBA object. It provides functions for the client to request and release a service, manages the status of applications and resources in this node and carries out the resource allocation

decision making. To keep the status of the services and the resources, the Node Inspector maintains three tables:

##### 1) Application Service Level Agreement table

This table is set up during the initialization. It keeps Service Level Agreement for each customer.

```

struct ASLAElement{
    long CustomerID;
    ServiceType st; };
typedef sequence<ASLAElement> ASLA;
  
```

##### 2) ResourceInfo table

This table tracks resource availability and busy information.

```

struct ResourceInfo {
    short DomainId;
    short ResourceId;
    // 1.switch 2. printer
    short AvailabilityInfo;
    //0 available, 1 not available
    short BusyInfo;
    // 0 not busy, 1 busy
    long LastUpdateTime;
    // last update time of switch
    string ResourceName;
    SwitchParameter SwitchParam; };
  
```

##### 3) Application information table

This table keeps track of each job including the customer, jobId, service type and Resource Inspector. The JobId is included because one customer may request several services.

```

struct ServiceParameter {
    long QoSType;
    //1.BDRJ, 2.level, 3.best effort
    QoSV QoSValue; };
struct ServiceType {
    short ResourceType;
    ServiceParameter ServiceParam; };
  
```

When the client issues an application request, the Node Inspector checks whether this is a valid customer ID and if so uses it to access the Application Service Level Agreement Table. If the request is allowed by this ASLA, the Node Inspector searches for the appropriate resource according to the system's status and the requirement, otherwise, the request is denied. The advantage of using the admission control is to enable the middleware to perform high-level request management in addition to the resource management.

```

void requestService(customerId,level,
resource parameters, out jobId, out RIGrant)
//return jobId:-1 denied by ASLA;
0 not available;
short releaseService(in long jobId)
//return 0 success, 1 fail
  
```

After admission control, the Node Inspector will perform the resource selection and optimization. The best resource for the request is determined and the job is carried out there.

##### 4.2. Resource inspector

The Resource Inspector is implemented as a standard CORBA object and is local to the resource. It provides functions for the resource to update status, to send the data to the resource, to manage the local resource status and job status.

The Resource Inspector maintains one table, RIJobInfo, with information on jobs assigned to its

corresponding resource and for the detailed status information of that resource. The resource specific parameters describe the details of the resource such as `SwitchParameter` for switches.

```
struct RIJobInfoElement{
    long CustomerId;
    long JobId;
    long ApplicationId; //ftp,www...
    SwitchParameter SwitchParam; };
```

#### 4.2.1. Resource interface

Resource interface is used for initialization, status updates, periodic “hello” messages and for reporting of failure to meet a QoS agreement. For example, if the resource is a switch, this interface is used whenever the total Bandwidth available exceeds a prescribed threshold or if the average delay exceeds the QoS agreement using the resource specific parameters.

```
updateRI(ResourceId,CustomerId,
    resource specific parameters)
```

#### 4.2.2. Event channel performance

Timing across the Event channel is critical. The latency introduced by the Event Channel ranges from 6 to 10 milliseconds. This is well within the time constraints of most application requirements. It does not increase significantly when more than one resource generates an event simultaneously. For additional information on the event channel performance see [7].

### 5. Conclusion and future work

We have presented an architecture as a middleware for networked applications and resource management protocols. Our system allows decisions that incorporate resource information across multiple domains, facilitating the application's task, enabling quality of service requirements to be met, and resulting in load balancing for a variety of resource types. The operations and mechanisms exploited here are application-transparent. Applications are only required to be informed of the Node Inspector's interface; they don't need to worry about where the object resides, what language the object is written in, what OS/hardware platform it runs on, or what communication protocols and networks are used to interconnect distributed objects.

#### References

- [1] P.Chandra, A.Fisher, C.Kosak, T.S.E. Ng, P.Steenkiste, E.Takahashi, H.Zhang, “Darwin: Resource Management for Value-Added Customizable Network Service”, *Sixth IEEE International Conference on Network Protocols (ICNP'98)*, Austin, October 1998.
- [2] P.Chandra, A.Fisher, P.Steenkiste, “A Signaling Protocol for Structured Resource Allocation”, *IEEE Infocom'99*, New York, March 1999.
- [3] P.Chandra, A.Fisher, C.Kosak, P.Steenkiste, “Network Support for Application-Oriented Quality of Service” *Sixth IEEE/IFIP International Workshop on Quality of Service*, Napa, May 98.
- [4] S.Chatterjee, J.Sydir, B.Sabata, “Modeling Application for Adaptive QoS-based Resource Management”, *Proceeding of the 2nd IEEE High Assurance Systems Engineering Workshop*, August 1997.
- [5] “DIRM: Distributed Integrated Resource Management, Technical Overview”  
<http://www.dist-systems.bbn.com/projects/DIRM>
- [6] P.Florissi, Y.Yemini, “Management of Application Quality of Service”, *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1994.
- [7] S.Knaiyar, V.Tsaoussidis, H.Badr, “Real-Time Application Management Based on CORBA Event Service” 17th International Conference on Applied Information, AI '99, pp.502-506, ACTA Press, Innsbruck, Austria, February 1999.
- [8] S.Mazumdar, Inter-Domain Management between CORBA and SNMP”, *Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Italy, October 1996.
- [9] S.McCanne, “Towards a Common Infrastructure for Multimedia-Networking Middleware”, *Proceedings of 7th Intl. Workshop on Network and Operating Systems for Digital Audio and Video (NOSSDAV'97)*, May 1997.
- [10] Object Management Group, “The Object Request Broker: Architecture and Specification”, Revision 2.0, July 1996.  
<http://www.omg.org>
- [11] Object Management Group, “CORBA Services: Common Object Services Specification, Trading Object Service Specification v1.0”, March 1997.
- [12] M.Ott, “What is wrong with QoS Research? ” *CCRL, NEC Internal Report.*, 98-N-002, 1998.
- [13] M.Ott, G. Michelitsch, D. Reininger, G. Welling, “An Architecture for Adaptive QoS and its Application to Multimedia Systems Design”, *Computer Communications special issue on Guiding QoS into Distributed Systems*, 1997.
- [14] D.Schmidt, “TAO: a High-performance ORB End system Architecture for Real-time CORBA”,  
<http://www.cs.wustl.edu/~schmidt/ACE-paper.html>
- [15] D.L.Tennenhouse, J.M.Smith, W.D.Sincoskie, J.Wetherall, and G.J. Minden, “A Survey of Active Network Research”, *IEEE Communications Magazine*, Vol.35, No.1, 1997.
- [16] Y. Tian, S. Frank, “Middleware Design Issues”, Technical Report 06.06.2000, Computer Science, SUNY at Stony Brook, June 2000.
- [17] V.Tsaoussidis, H.Badr, G.Sazaklis, “Application-oriented Cross-Domain Resource Management Schema using CORBA”, *IEEE Internet Workshop '99, IEEE IWS '99*, pp. 53-60, IEEE Press, Osaka, Japan, February 1999.
- [18] V.Tsaoussidis, A.Deka, C.Comaniciu, C.Kulikowski, “Application-oriented System for Quality of Service (ASQoS)”, *IEEE SICON '98*, Singapore, July 1998.