Using CORBAís Advanced Services to Enhance the Integrity of QoS Management Programmable Networks

Qiang Gu and Alan Marshall

Advanced Telecommunications System Laboratory,
School of Electrical & Electronic Engineering,
The Queenís University of Belfast,
Ashby Building, Stranmillis Road, BT9 5AH Belfast
Northern Ireland, U.K.
{qiang.gu, a.marshall}@ee.qub.ac.uk

Abstract. The development of wideband network services and the new network infrastructures to support them have placed much more requirements on current network management systems. Issues such as scalability, integrity and interoperability have become more important. Existing management systems are not flexible enough to support the provision of Quality of Service (QoS) in these dynamic environments. The concept of Programmable Networks has been proposed to address these requirements. Within this framework, CORBA is regarded as a middleware technology that can enable interoperation among the distributed entities founds in Programmable Networks. By using the basic CORBA environment in a heterogeneous network environment, a network manager is able to control remote Network Elements (NEs) in the same way it controls its local resources. Using this approach both the flexibility and intelligence of the overall network management can be improved. This paper proposes the use of two advanced features of CORBA to enhance the QoS management in a Programmable Network environment. The Transaction Service can be used to manage a set of tasks, whenever the management of elements in a network is correlated; and the Concurrency Service can be used to coordinate multiple accesses on the same network resources. It is also shown in this paper that proper use of CORBA can largely reduce the development and administration of network management applications.

1 Introduction

The provision of QoS-guaranteed network services on-demand requires a more advanced network management architecture. The use of programmable network architectures is a new approach aimed at increasing the flexibility and manageability of networks [6],[7],[8]. CORBA from the Open Management Group (OMG) provides an enabling environment with many useful facilities that can be applied to these architectures. In this paper, two of CORBAís services, the Transaction Service and the Concurrency Service, are used to improve the integrity of the network management system.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-45812-8_28

Sections 2, 3 and 4 give an overview of QoS, programmable network architectures and the CORBA standard respectively. Section 5 describes the problem with managing network elements that are both distributed and correlated. In section 6, a new management architecture based on CORBAís Transaction Service is proposed to address the problem. A case study with this architecture is given in section 7. Section 8 investigates CORBAís Concurrency Service, which can be used in multi-access scenarios. With regard to the common concern of CORBAís performance and other new network management architecture (i.e. WBEM), the authors have provided a discussion in section 9. Section 10 provides the conclusions of this work.

2 QoS Architectures

There is more than one way to characterize Quality of Service. Generally, QoS is the ability of a NE (an application, host, switch or router) to provide some level of assurance for consistent network delivery of specific network connections. Applications, network topology and policy dictate which type of QoS architecture is most appropriate for individual or aggregates traffic flows (connections). The evolution of IP networks from their current best-effort delivery of datagrams into guaranteed delivery of time sensitive services is still far from complete. Resource Reservation (RSVP)[15],[16] and Differentiated Services (Diffserv) [3] have been proposed as mechanisms for implementing Service Quality in IP based networks.

One apparent issue is that it is impossible to use just one router or switch in the network to achieve the guaranteed service. All the NEs will need to conform to the same configuration scheme and work cooperatively. For example, the end users only care about the end-to-end performance of a network application. They may have no idea of how many routers are being used to carry the traffic. But from network providerís point of view, in order to provide such a guaranteed service, all the routers along the path of an application must obey the same rules. If any of them fail to do so then it is impossible to achieve the guaranteed end-to-end service.

3 Programmable Network Architectures

The NEs in present day networks are mostly vertically integrated closed systems whose functions are rigidly programmed into the embedded software and hardware by the vendor. The paradigm 'Programmable Network' envisions the future telecommunications network as an open system that can be programmed to deliver QoS based network services e.g. voice, video and data in a customisable manner.

The framework considered here will be based on the emerging standard for open interfaces for Programmable Network equipment, namely the IEEEís P1520 reference model [10]. Figure 1 shows the architecture of this model. To date, most of the research in this area has tended to concentrate on the network generic services layer (between the U and L interfaces). An important feature of the research considered here, is the mapping of higher layer requirements onto actual physical resources in the network equipment. The current P1520 model considers that all the legacy networkís resources (e.g. routing, scheduling, PHB etc) are abstracted into objects. However, P1520 has only defined the basic access APIs on network resources. There is not any definition for advanced business logics in P1520. Features such as transaction, concurrency and log are commonly used in centralized system. By using CORBAís rich set of services, programmable network architectures can also benefit from these features.

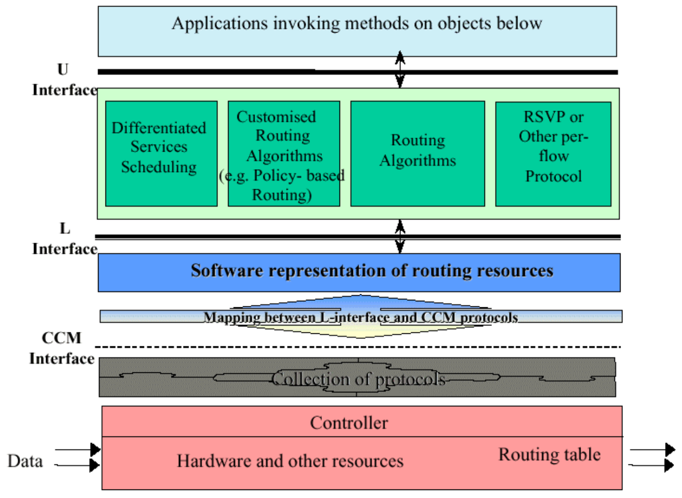
Applications invoking methods on objects below
U Interface
Differentiated Services Scheduling
Customised Routing Algorithms (e.g. Policy- based Routing)
Routing Algorithms
RSVP or Other per-flow Protocol
L Interface
Software representation of routing resources
Mapping between L-interface and CCM protocols
CCM Interface
Collection of protocols
Controller
Data
Hardware and other resources
Routing table

Fig. 1 IEEE P1520 Architecture [10]

4 CORBA as a Standard Middleware Solution

The Common Object Request Broker Architecture (CORBA) from Object Management Group (OMG) has been widely considered as the choice architecture for the next generation of network management [1],[2]. Until now, most research has focused on using CORBA to integrate incompatible legacy network management systems such as SNMP and CMIP [11], and the definition of specific CORBA services (e.g. Notification, Log). In addition to the basic features of CORBA, there are also some other CORBA services, which can be very useful for network management applications. In this paper, the use of the Transaction Service and Concurrency Service to achieve better QoS control is investigated.

It is widely accepted that transactions are the key to constructing reliable distributed applications. The CORBA Transaction Service supports the concept of transactions as atomic executions. In a distributed environment, these executions can span more than one NE. By either committing or rolling back, a transaction can always maintain a consistent state in a designated NE. The Transaction Service defines the interfaces that allow multiple and distributed objects to cooperate to provide atomicity. The Service provides transaction synchronization across the elements of a distributed client/server application. Its functionality includes: controlling the scope and duration of a transaction; allowing multiple objects to be involved in a single, atomic transaction; allowing objects to associate changes in their internal state with a transaction; and coordinating the completion of transactions.

The purpose of the Concurrency Control Service is to mediate concurrent access to an object such that the consistency of the object is not compromised when accessed by concurrently executing computations. In the context of network management, this can be access to a network resource by more than one distributed application. The Concurrency Control Service consists of multiple interfaces that support both transactional and non-transactional modes of operation. The user of the Concurrency Control Service can choose to acquire a lock on resources in one of two ways:

1. On behalf of a transaction (transactional mode.) The Transaction Service implicitly acquires and releases the locks on a resource. After obtaining the lock, the client can access the resource exclusively. This prevents other users changing the same resource which may corrupt its integrity.
2. By acquiring locks on behalf of the current thread (that must be executing outside the scope of a transaction). In this non-transactional mode, the responsibility for dropping locks at the appropriate time lies with the user of the Concurrency Control Service

5 Management of Correlated NEs

Two terms are used in this paper. A **management session** is one single interoperation between a network manager and a NE. Additionally, a **management task** is more user-oriented. A task should have a specific significance for the end users (i.e. achieve a meaningful function). The following example shows that sometimes, a task will have more than one session involved. Figure 2 shows a case study network. In this network, there are three video conferencing applications running between 3 pairs of server/clients. A network formed by router1 through router3 connects these applications. There are two bottlenecks in the network. One is the link between router1 and router2. The other is the link between router2 and router3. The three applications have the same traffic profile, labeled as traffic1, traffic2 and traffic3. Traffic *n* is the traffic between video_server_*n* and video_client_*n*. All the routers use the same criteria to classify the traffic, and Weighted Fair Queuing (WFQ) is used in each router to schedule the traffic for each application.

Initially, the weights for each type of traffic are the same in every router, which means they receive the same service from network. Because they share the same traffic profile, these applications have the same throughput and delay. If for some reason, the network administrator wants to give a certain flow (e.g. traffic2) lower delay, then this can be done by increasing the weight for that flow in the routers it passes through. In this test network, because the transmission and propagation delays for all the traffic are the same, end-to-end delay will mainly depend on the queuing delay in each router along the path.

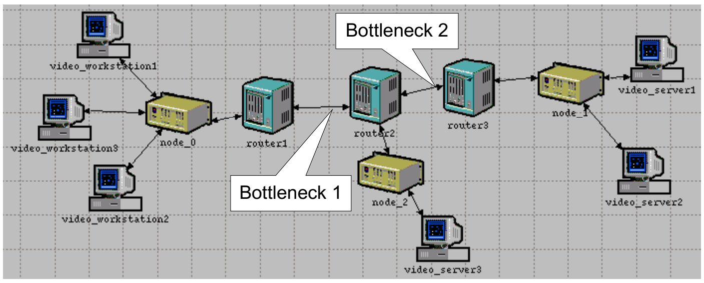

Fig. 2 Case Study Network

In this scenario, only the weight for traffic2 on router3 was increased. Traffic2 has lower queuing delay in router3. However, traffic2ís queuing delay is increased in router2. As the result, there is no significant end-to-end delay improvement for traffic2.

The explanation for this phenomenon is that by increasing the weight for traffic2 on router 3, more traffic2 is forwarded to the downstream router2. Because the WFQ on router 2 still has its original weight, an increase in the delay is inevitable. In this example, the end-to-end delay is the meaningful performance statistics for the end users. Merely changing the weight on one router cannot achieve the desirable effect.

The example shows that some management tasks will need to have more than one NE involved. Every management session on an individual NE is only a part of the overall management task. The task may not be fulfilled if a session in the chain is not executed properly. It is therefore necessary that the manager be informed if a session failed. The manager can then take appropriate action to reverse the change of other successful sessions in the same task. The integrity of network is hence guaranteed. Because the manager and NEs are in different locations, there needs to be distributed coordination.

It can be expected that as the network scale and complexity increase, the synchronization of NEs will become more and more difficult. The services may be very dynamic. The provision of service may only span a relatively shorter term. This makes the network configuration in a very changeable. The network integrity is apt to be violated. As the result, no services can be guaranteed.

As well as the synchronization of data across NEs, it is also necessary to synchronize data across different network managers in the overall end-to-end connection. There will be more and more interactions between the sub-managers in a large network where topology and management relationships can vary quite dramatically. Therefore if one manager wants to affect the behavior of a number of others, the originator should be able to guarantee that all the changes affected on the other managers can be committed or rolled back as a whole. For example, a higher-level manager may wish to change the billing information on all its sub-managers. If only some have successfully updated to the new price schemes, unfairness may occur between clients residing in different management domains. To avoid this, the Transaction Service can be used to make sure all the customers receive the same price scheme.

The concept of distributed transaction is not available in either SNMP or CMIP. It is left to the system developer to make a management application transactional. This approach has several disadvantages. Firstly, without a standard distributed transactional mechanism, it is impossible to achieve interoperability in heterogeneous environments. Secondly, the management application developers need to write their own code to support distributed transaction, which is can be a very complex and error-prone task.

6 Using CORBAís Transaction Service to Enhance Network Management in a Distributed Environment

Figure 3 shows the architecture of using the CORBA Transaction Service in network management. In order to deploy it, there must be at least one CORBA server in the network with the Transaction Server running on it. The manager with Transaction-Service-compatible ORB installed is the client or, in the terms of Transaction Service, the originator. It can choose the participant NEs and the options of a transaction. During the transaction, the manager will invoke the method on all the other participants. At the beginning of the transaction, the participants will register their resources involved with the Transaction Server. The Transaction Server will then coordinate the participants during the transaction. All the participants take part in the two-phase commit protocol. If for any reason the change on one NEís resource fails, the NE can report the failure to the Transaction Server. This allows the Transaction Server to roll back all the changes on other transaction participants. Otherwise the transaction server can conduct all the participants to commit the changes. In either case, the integrity of the network is maintained.

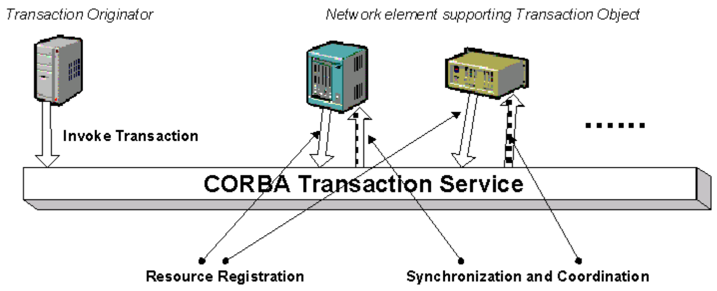
Transaction Originator
Network element supporting Transaction Object
Invoke Transaction
......
CORBA Transaction Service
Resource Registration
Synchronization and Coordination

Fig. 3 Using CORBA Transaction Service in Network Management

Figure 4 illustrates an example where the CORBA Transaction Service is used to change the WFQ tables on all the routers along a path. In this example, the incoming packets to each router are classified according to certain criteria in each router. If the network provider wants to allocate a certain amount of bandwidth to a specific class of traffic along a path, then the network manager needs to contact all the routers along this path and change the settings on each of them. The network manager can invoke the appropriate methods on these routersí WFQ entities to change the classifying criteria and the bandwidth allocated for incoming packets.

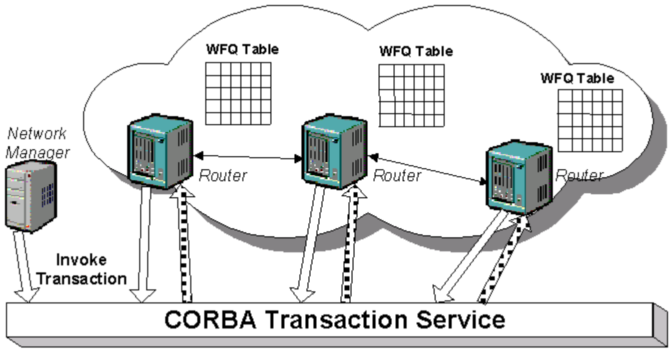

Fig. 4 Using CORBAís Transaction Service for Queue Configuration

To make the WFQ entities in each router accessible, they need to be wrapped into a CORBA object. Following code shows an example IDL code of such a queuing system.

```
// QueuingSystem.idl
#include "CosTransactions.idl"
module QueuingSystem{
  typedef unsigned long IP_ADDRESS;
  interface QueueProfile  {
    readonly attribute unsigned long queue_profile_id;
    attribute IP_ADDRESS source_addr;
    boolean queue_equ (in QueueProfile
in_queue_profile);
    // can be extended to support more complicate queue
classification
  };
  interface Queue  {
    readonly attribute unsigned long queue_id;
    attribute QueueProfile  queue_profile;
    attribute unsigned long weight;
    attribute float band_width;
  };
  typedef sequence<Queue> QUEUE_LIST;
  interface Queue_Sys: CosTransac-
tions::TransactionalObject,CosTransactions::Resource  {
    attribute unsigned long total_weight;
    attribute QUEUE_LIST queues;
    void add_queue_weight(in QueueProfile
in_queue_profile, in unsigned long weight);
    void add_queue_bandwidth(in QueueProfile
in_queue_profile, in float band_width);
    void remove_queue(in QueueProfile
in_queue_profile);
    void modify_queue_weight(in QueueProfile
in_queue_profile, in unsigned long new_weight);
```

```
     void modify_queue_bandwidth(in QueueProfile
in_queue_profile, in float new_bandwidth);
   };
};
// QueuingSystem.idl
```

The interface *QueueProfile* is used to specify the classification criteria. The interface *Queue* is used to specify the setting for a certain queue. The *Queue_Sys* is the interface that can be accessed to change all the configuration of queuing system (add a queue, remove a queue etc.). *CosTransactions::TransactionalObject* is the super class of all the objects, which need to be transactional.

7 Case Study

The test network described in Section 5 was used to study how to use the Transaction Service to enhance the QoS management in the Programmable Network. Two scenarios are compared. In the first the manager changes the weights of traffic2 on router3 and router2 without a Transaction Service. The weight on router3 is changed at 60 seconds and the weight on router2 is changed at 90 seconds. In the second scenario, the manager performs the same task via the Transaction Service. Figures 5 and 6 show the results of two scenarios respectively. It can be seen that in scenario 1, the end-to-end delay will only change after the weight for traffic2 on router2 was increased (at 90 seconds). While in scenario 2, the end-to-end delay for traffic2 makes a significant improvement after just 60 second.

It should be noted that in order to illustrate the difference between the transaction and non-transaction scenarios, the weight on router2 was deliberately changed in the first scenario. In the real network applications, this delay can be much shorter. However, when there are many NEs in a path, it will take very long time to do all the changes.

8 Using the CORBA Concurrency Service to Enhance Network Management in Distributed Environments

Concurrency is not a significant issue when most of the time, only one network manager would access a network element at a certain time point. The emergence of programmable network has greatly changed the above assumption. In order to provide more customizable services, the network providers may open some resources on network devices to third party providers or even end users. There will always be the possibility of more than one thread running simultaneously wishing to manipulate the same resource. There are two solutions for this problem:

- Every network element has a build-in mutual exclusion (mutex) mechanism to protect critical data
- Employ a third party concurrency service to serialize the access to the critical data in the network elements.

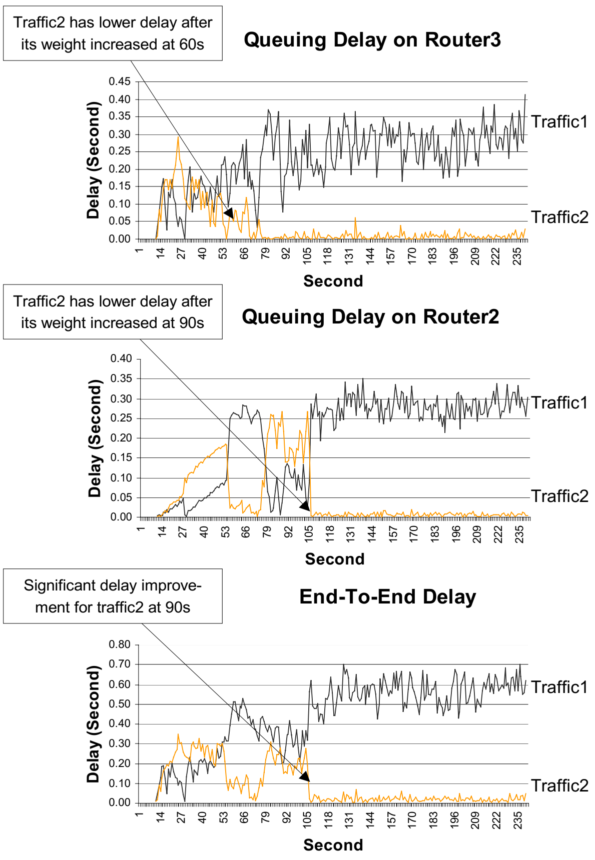

Fig. 5 Result of Changing Weight without Transaction Service

One obvious disadvantage of the first solution is that it requires that all the network elements upgrade their software to support concurrency. To achieve the complicated mutex mechanism in a network element with limited memory and CPU power might result in a significant decrease in its performance. By using a third party concurrency service in the network, all the network elements can continue to operate as before. They do not even need to know that there is some other service in the network coordinating the access to the critical data on themselves. These concurrency services can be run on a generic hardware and software platform with adequate memory and processing power. Such a platform (e.g. a workstation) can be the proxy for many network elements. This provides a very flexible approach to configuring the number, location and load of the concurrency services in the network.

Another potential disadvantage of the first approach is the incompatibility between different vendors and platforms. The same code may not be able to access the critical data in different devices because they have different concurrency APIís. This problem will not occur in the second approach, whenever only one standard concurrency service API is used across the whole network.

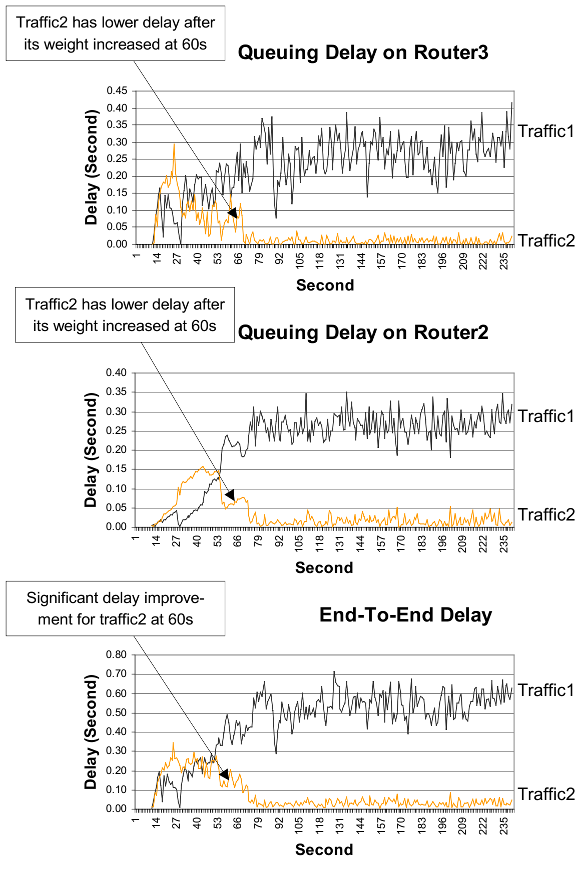

Fig. 6 Result of Changing Weight with Transaction Service

In this paper, CORBAís Concurrency Service is investigated as one choice for the second solution.

The use of CORBAís Concurrency service is very straightforward. Figure 7 shows its general application in network management. There are normally three steps. In step 1, the network element creates the *LockSet* on CORBA Concurrency server and associates it with the internal resource. There is no limit on the number of *LockSets* a network element can create. One Concurrency server can also serve any number of network elements. In step 2, if a client wants to access a certain network resource, it will firstly need to acquire the *LockSet* for that resource. If there are multiple accesses to a resource, the Concurrency Server will coordinate all of them. The Concurrency Server serves the requests and grant *LockSets* according to the mutual exclusion rule. In step 3, only the client that obtains the *LockSet* can then access the real network element resource. The others must wait for the release of *LockSet*.

The use of transaction and concurrency also raises the issue of deadlock within network nodes. This becomes exacerbated whenever the management system is dis-

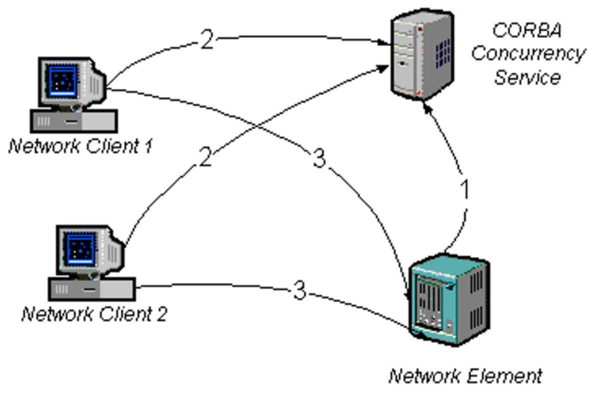

Fig. 7 Using CORBAís Concurrency Service In Network Management

tributed. There are however, a number of solutions to for it (e.g. deadlock prevention, deadlock avoidance, deadlock detection). CORBAís Concurrency Service also has a built-in deadlock avoidance mechanism ñ the *upgrade mode lock*. In addition to common read or write mode lock, an upgrade mode lock is a read lock that conflicts with itself. It is useful for avoiding a common form of deadlock that occurs when two or more clients attempt to read and then update the same resource. If more than one client holds a read lock on the resource, a deadlock will occur as soon as one of the clientsí requests a write lock on the resource. If each client requests a single upgrade lock followed by a write lock, this deadlock will not occur.

9 Discussion

One common concern of CORBA-based applications is their complexity. CORBAís advanced services may be viewed as yet another sophisticated component in an already complex management system. It is very likely that the management system with CORBAís advanced Service will have higher overall complexity. However, compared to traditional network management architectures, this new approach has some advantages:

✄ In the traditional management system, almost all the intelligence is placed in the central manager. The NEs are more like dumb devices. Most decisions are made by the manager and sent to NEs. When the scale and complexity of network management applications increases, the manager will find itself in a very overloaded state. The development of embedded technologies (e.g. embedded JAVA, embedded CORBA) has made it possible to use a distributed computing approach to achieve the same functionality but with an alternative architecture. The idea of distributed computing is to spread the intelligence and load into the entire network. This is based on several assumptions. Firstly, it is cheaper to use multiple NEs to achieve the functionality of one single mainframe manager. Secondly, the NEs may have more knowledge of local environment and thus can have quicker and more accurate response. In the case of CORBAís Transaction and

Concurrency Services, the NEs installed with embedded ORBs will take a more active role in the management. With this approach the manager has much less functions to perform. The only thing the manager needs to do is to trigger the transaction at the beginning and wait for the result (e.g. commit or rollback) when it is finished. If the scale and the complexity of a network are increased, the extra load will be shared by newly added NEs. The managerís load will be maintained at an almost constant level.

- From the developersí point of view, to develop a distributed transaction or concurrency application can be a complex task. CORBA allows distributed application developers to enjoy the elegance of OO methodology. By using third party standard services, a large portion of development effort can be saved.
- One significant feature of CORBA is the reusability. It is very common that a protocol has more than one versions or releases. The early versions may not be able to satisfy all the requirements of a network application or may not predict the future requirements so some revised versions will be issued later. It is desirable that the development of new versions or new releases can use the work from old version or release as much as possible. By using the inheritance mechanism, the work in an old version or release can be easily reused.

In the development and deployment of network applications, there are two kinds of timescales. One is used to measure how fast an application can fulfill a particular service requirement (e.g. using an algorithm to manage the current network infrastructure and then provide end to end delay-constrained virtual links). The other is used to measure how fast an application with new service requirements can be developed and deployed into the current network environment. OMG has defined the standard for embedded and real-time CORBA. There are already some real-time CORBA implementations available. Regarding the timer of developing a new application, CORBA has undoubted superiority over the traditional protocols. And providing new services and applications to satisfy usersí demand as soon as possible is just the key to win the competition in the current marketplace.

Recently, Web-Based Enterprise Management (WBEM) [13] and Common Information Model (CIM) [14] from Distributed Management Task Force (DMTF) is another trend of network management. Comparing to CORBA-Based architecture, both of them have their advantages and disadvantages.

WBEM uses XML and HTTP as transport encoding and protocol. Because of the popularity of HTTP in PC and Unix workstation, it is easier to add WBEM into enterprise where general workstations are the majority of network elements. Sunís Java Management eXtension (JMX) and Microsoftís Windows Mgmt Instrumentation (WMI) are all based on WBEM. However, it should be noted that fully functioned and extensible HTTP server is still not common in network devices (e.g. router, switch etc.). Another problem of HTTP is that it can only run on TCP/IP. CORBA uses GIOP as transport protocol. As a middleware, it is designed to be independent from underlying physical network protocol. This feature makes CORBA more flexible.

CIM is an approach to the management of systems and networks that applies the basic structuring and conceptualization techniques of the object-oriented paradigm. A management schema is provided to establish a common conceptual framework. The management schema is divided into these conceptual layers: core, common and extension. The language used to define CIM (e.g. MOF, XML) is the equivalence of IDL. The CIM schema is similar to the core classes in CORBA. The elegancy of IDL is that it is independent of any programmable language. As the same time, by using IDL compiler, the IDL code can be translated into any specific language and integrated with current code.

CORBA uses binary encoding for all the information. It is not friendly for end users. But it is easy for machine to manipulate the data. On the contrary, XML uses textual encoding. XML parsing is always necessary. So generally, information decoding of CORBA is more efficient.

10 Conclusion

This paper describes the use of CORBA's Transaction Service and Concurrency Service to enhance the management in Programmable Network environment. While it is very common that these two services are used together in an IT domain (e.g. enterprise database), this paper has shown that they can also be employed in the telecommunications network management domain, where they can cooperate to achieve better integrity and consistency than conventional network management techniques. The paper specifically describes how these advanced CORBA services can be deployed in future heterogeneous open and Programmable Networking environments in order to perform distributed management tasks such as configuring NEs to meet QoS targets.

Acknowledgements

The authors would like to thank the Industrial Research and Technology Unit (IRTU) of Northern Ireland for their financial support for the research carried out in this work. The technical support from NITEC (Northern Ireland Telecommunications Engineering Centre, of Nortel Networks) is also gracefully acknowledged.

References

1. Paul Haggerty and Krishnan Seetharaman The benefits of CORBA-based network management Communications of the ACM, 41(10), October 1998, pp. 73-79.
2. Luca Deri Network Management for the 90s ECOOP'96 Workshop on 'OO Technologies For Network and Service Management' Proceedings, July 1996.
3. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. An Architecture for Differentiated Service. December 1998.
4. QoS Protocols & Architectures, QoS Forum white paper, 1999 http://www.qosforum.com/white-papers/qosprot_v3.pdf
5. Document formal/01-05-02 (Object Transaction Service specification, v1.2), OMG

6. David L. Tennenhouse et al. A survey of active network research, IEEE Communications Magazine, pages 80-86, January 1997.
7. Andrew T. Campbell et al. A Survey of Programmable Networks, ACM Computer Communications Review, April 1999.
8. David L. Tennenhouse and David. J. Wetherall, Towards an Active Network Architecture Computer Communication Review, Vol. 26, No. 2, April 1996.
9. IEEE P1520 Programming Interfaces for IP Routers and Switches, an Architectural Framework Document.
 http://www.ieee-pin.org/doc/draft_docs/IP/p1520tsip003-04dec98.pdf
10. Jit Biswas et. al., ìIEEE P1520 Standards Initiative for Programmable Network Interfacesî, IEEE Communications Magazine, Vol.36, No.10, October 1998, pp. 64-70.
11. OMG CORBA/TMN Interworking Specification, version 1.0
 http://www.omg.org/cgi-bin/doc?formal/2000-08-01
12. Document formal/00-06-14 (Concurrency Service stand-alone document), OMG
13. Web-Based Enterprise Management (WBEM) Initiative
 http://www.dmtf.org/standards/standard_wbem.php
14. CIM Specification v2.2
 http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf
15. R. Braden, D. Clark, S. Shenker Integrated Services in the Internet Architecture: an Overview.. June 1994.
16. R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification.. September 1997