碩士學位論文

# OSI 관리 방식의 CORBA 트래픽 모니터링을 위한 프록시 설계



濟州大學校 大學院
情報工學科

朴 宰 成

1998年 12月

# OSI 관리 방식의 CORBA 트래픽 모니터링을 위한 프록시 설계

指導敎授 宋 旺 暾

朴 宰 成



이 論文을 工學 碩士學位 論文으로 提出함

1998年 12月

朴宰成의 工學 碩士學位 論文을 認准함

審査委員長 _____ 印

委 員 _____ 印

委 員 _____ 印

濟州大學校 大學院

1998年 12月

# The Design of Proxy for CORBA Traffic Monitoring through OSI Management Method

JaeSeong Park



(Supervised by professor WangCheol Song)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INFORMATION ENGINEERING
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

1998. 12.

# 목 차

## Summary

System monitoring is a part of the system management. It is a vital function for the proper operation of a system in use. Currently, OMG is studying CORBA system management in the purpose of efficient control of CORBA system itself. However, with no certified standards in hand yet. However, many companies and research laboratories have been developing various kinds of CORBA monitoring tools to obtain information about CORBA traffic. Their tools are providing only to a specific ORB.

In this paper, I study on the traffic monitoring for CORBA resources through OSI management method. In the first place, I design a proxy to monitor the CORBA traffic and then transfer the traffic information to MO. Secondly, I define MO to store the traffic information and to build MIB. The monitoring system consists of a CORBA server, a proxy object, and a MIB. The CORBA server is made up of a service provider, a proxy server, and an event server. The service provider providing general CORBA services. The proxy server gathers traffic information. When exception happens, a event server sends its information to a proxy object. The proxy object acts as a process of a CORBA object. It`s made up of a proxy client, an event client, and an IPC server. The proxy client gets data from server object. The event client receives the exception data. The IPC server sends data to MOs.

# I. 서론

현재의 네트워크는 점점 거대해지고, 지역적으로 멀리 떨어진 곳일지라도 인터네트를 경유하면 어느 곳이든지 쉽고 빠르게 연결할 수 있게 되었다. 이렇듯 점점 커지고 고속화 되는 네트워크를 효율적으로 사용하고 하는 방법 중에 하나가 클라이언트/서버 방식이다. 그러나 이런 시스템에서도 한계를 가져와 좀더 발전적인 형태의 시스템인 분산 시스템이 대두되고 있다. 특히 이런 분산 시스템 중에서 현재 가장 많이 주목을 받고 있는 기술이 CORBA(Common Object Request Broker Architecture)이다. 현재 OMG(Object Management Group)(1989)에서 표준화 작업을 하고 있다. CORBA는 분산 시스템에서 객체라는 개념이 도입이 된 형태로 객체지향의 네트워크 연결을 제공하는 것이라 할 수 있다.



시스템 모니터링은 시스템의 특성이나 정보를 얻기 위한 부분으로 시스템의 성능을 파악하는 것 외에도 최적의 환경을 구성하거나 관리하기 위한 시스템 관련 정보를 얻는데 있다. 그러나, OMG(Object Management Group)에서 이러한 시스템 관리 부분의 표준화가 만족스러운 수준이 되지 못하고 있다(OMG,1995). 현재 기본적인 표준안이 제안되어 있을 뿐이고, 벤더들이 이를 가지고 구현하기에는 부족하기 때문에 더 많은 연구가 필요하다(MAScOTTE,1998). 그러나 아직 까지는 시스템 관리 부분에 있어서 표준화된 표준안이 없기 때문에 각 벤더에서 개발한 관리 프로그램들은 자신들이 제공하는 서비스 등을 통해서 제공되고 있다(Inprise,1997, Iona,1997,

BLACK&WHITE,1992). 더욱이 이런 시스템 관리 프로그램은 특정 ORB(Object Request Broker)에 국한 된다는 단점이 생긴다. 본 논문에서는 이런 단점을 극복하고자 OSI 관리 방법을 이용하여 CORBA 트래픽 모니터링을 하기위한 프록시를 설계 하였다.

본 논문의 2장에서 CORBA대한 이해와 GIOP/IIOP에 대해 논할 것이다. 3장에서는 OSI관리에 대해 살펴보고, 4장에서는 트래픽 모니터링을 위한 시스템 설계, 5장에서는 시스템 구현, 6장에서는 결론에 대해 살펴보기로 하겠다.

# II. CORBA

분산 시스템을 표준화하는 컨소시엄 중에 OMG에서 많은 발전이 이루어지고 있다. OMG의 CORBA는 객체 지향 기술의 캡슐화로 사용자로부터 서버의 구현을 은폐하지만, 분산 객체 기술은 네트워크 프로토콜과 네트워크상의 서버의 위치도 사용자로부터 은폐한다. 또한, 기존 응용프로그램을 분산 객체로서 래핑하는 것에 의해 기존 응용의 활용과 자연스러운 이전을 실현하고 있다. 그리고, OMG에서는 특정 네트워크 아키텍처나 OS 아키텍처에 의존하지 않고, 보다 추상화 레벨이 높은 단일의 아키텍처를 제공하는 것에 의해 이식성과 상호 운용성을 실현하고 있다(Onazawa,1997).



## 1. 참조 모델

분산 객체 환경을 구성하고 참조하기위해 OMG에서는 컴포넌트를 구성하고, OMG의 목표를 달성하기 위한 모델이 레퍼런스 모델이다. 레퍼런스 모델은 OMA(Object Management Architecture)를 구성하는 각각의 컴포넌트의 기능과 상호관계를 확실히 하고, 표준화 작업에 대한 프레임워크를 제시하고 있다. Fig.1에서 보이듯이 4개의 컴포넌트로 구성되어 있고, 이들 간의 관계를 보여주고 있다.

ORB는 객체간의 요구와 응답의 전송을 제공하는 메시징 기능을 제공한다. 공통 객체 서비스는 객체를 구현하기 유지하기 위한 여러 가지 서비스를 제공하기 위한 집합이다. 공통 퍼실리티는 응용프로그램에서 이용되는 기능을 제공하는 클래스 객체의 집합이다. 마지막으로, 응용프로그램 객체는 사용자 응용프로그램 고유의 오브젝트를 말한다. 특히, ORB는 이들 객체간의 통신을 담당하는 중요한 부분이다.

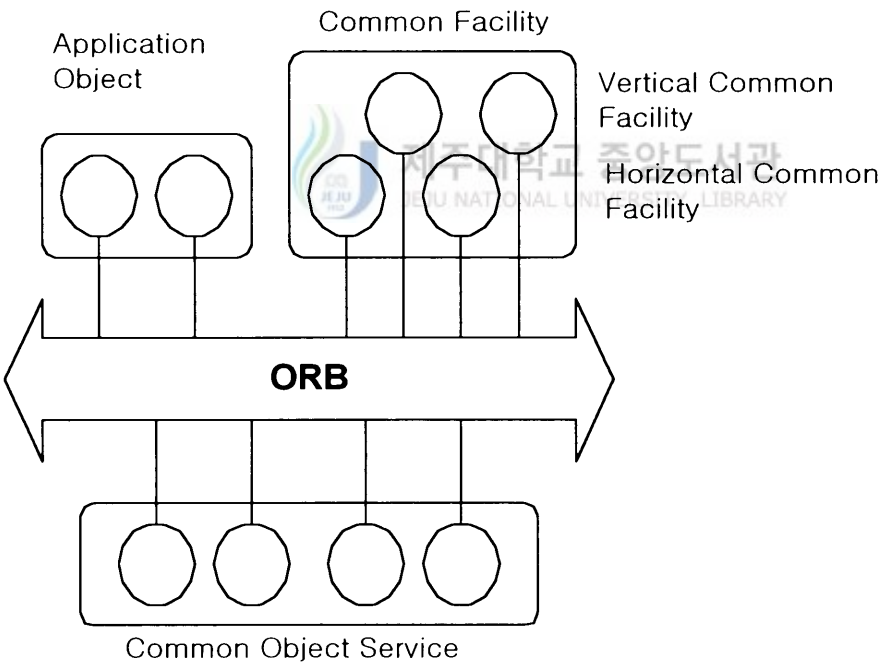
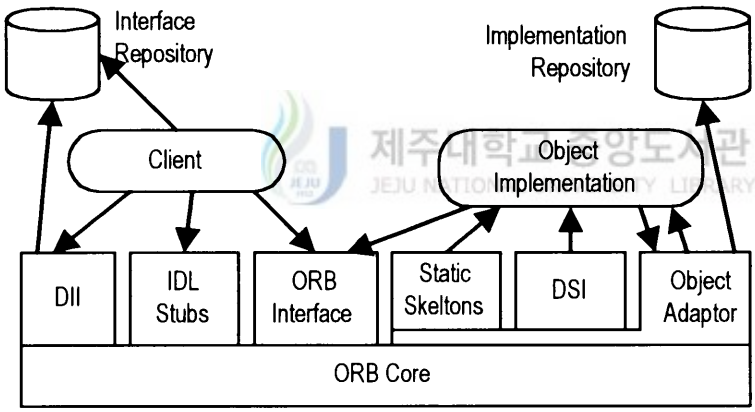



Fig. 1 OMG object management architecture

## 2. ORB의 구성

ORB는 CORBA 클라이언트에서 CORBA 서버로 메시지를 넘겨주거나, 반대의 역할을 한다. 이 ORB는 Fig.2에서 보여주듯이 ORB 코어, 객체 어댑터, ORB 인터페이스, 동적 기동 인터페이스, 동적 스켈레톤 인터페이스, IDL(Interface Definition Language) 스터브, IDL 스켈레톤, 인터페이스 저장소 및 구현 저장소로 구성된다.




DII: Dynamic Interface Invocation
DSI: Dynamic Skeleton Invocation
ORB: Object Request Broker
IDL: Interface Definition Language

Fig. 2 CORBA frame work

클라이언트는 객체 구현으로 특정 서비스를 요청을 한다. 객체 구현은 서비스를 제공하는 것으로 클라이언트에게 원하는 서비스를 제공하게 된다. 구현 저장소는 CORBA에서 참조 정보가 있는 곳으로 객체 구현에 관한 정보를 저장하고 있다. 인터페이스 저장소는 객체에 대한 인터페이스 정보

등을 저장하고 있다. 이들 대 부분이 벤더의 실장에 의존하고 있다. 가운데 있는 ORB 인터페이스는 ORB에 있는 기능을 사용하기위한 인터페이스를 제공한다. 마지막으로 객체 어댑터 부분으로 적절한 객체 구현으로 메시지를 넘기는 디스패칭 기능 외에도 객체의 생성과 삭제, 활성화와 비활성화, 예외처리 등을 행한다. 나머지는 객체와 ORB사이에서 메시지 전달을 담당하고 있다.

ORB에서 메시지의 흐름을 살펴 보면, Fig.3과 같은 형태로 이루어진다. 클라이언트는 요구 메시지를 객체 구현 쪽으로 발행해서 결과를 얻어오게 된다. 이때, 클라이언트와 서버는 ORB를 거쳐서 메시지는 전송하게 된다. 클라이언트는 메시지를 DII(Dynamic Interface Invocation)이나 IDL 스터브를 통해 ORB로 보내면, ORB는 서버측으로 메시지를 넘겨주게 된다. 그러면, 서버측에서 객체 어댑터를 거쳐서 정적 스켈레톤이나 DSI(Dynamic Skeleton Interface)통해서 객체 구현으로 메시지를 넘겨주게 된다. 서버에서 응답은 앞의 전송 순서에 반대 방향으로 이루어지게 된다.



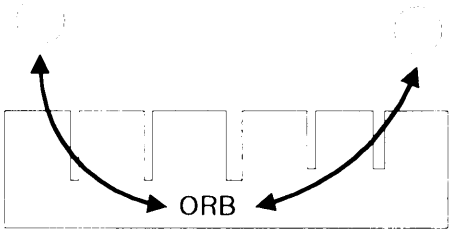



Fig. 3 The flow of message

## 3. GIOP

본 논문은 CORBA 가운데에서 GIOP부분을 중점적으로 다루었다. CORBA에서 메시지를 주고 받는 형태를 정해 놓은 부분이 GIOP이다. 즉, Fig.4에서 보면 GIOP는 ORB간 상호동작을 위한 프로토콜로 연결 지향 전송 프로토콜 상에 매핑되어 있다. 각각의 클라이언트에서 발생하는 메시지는 GIOP에서 정한 메시지 타입에 맞추어서 ORB상으로 보내지게 된다. 이를 IIOP라 하고, TCP/IP상에 매핑되어 있다(OMG,1998, Rovert Orfali,Dan Harkey,1998).



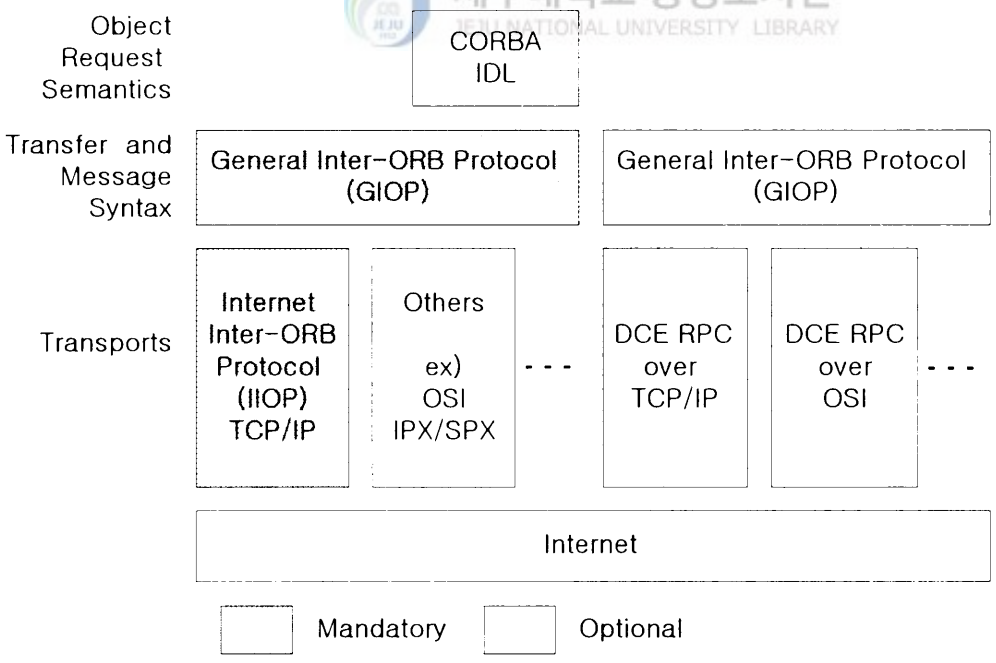



Fig. 4 The CORBA 2.0 inter-ORB architecture

### 3.1 GIOP 메시지 헤더

본 논문에서는 트래픽 데이터를 메시지로부터 추출하려고 한다. 이런 메시지 포맷이 기술된 곳이 GIOP로 이것에 대해 살펴보도록 하자. 먼저 GIOP 메시지 헤더 부분을 분석해보자. Fig.5에서 보면, 먼저 GIOP 메시지 임을 표시하는 **Magic**이라는 부분으로 GIOP 메시지를 판별한다. 이 곳에는 4 바이트 크기의 "GIOP"라는 글자가 대문자 형태로 인코딩되어 들어가 있다. 그리고, GIOP 메시지 헤더 부분에 **Version**부분에는 버전에 관한 정보가 들어가 있다. GIOP는 v1.1과 v1.0인 2 가지 종류를 가지고 있다. 이는 CORBA의 버전과 무관하고, CORBA 2.0에서는 GIOP 1.0과 1.1를 사용할 수 있다. **Byte-order**는 바이트 정렬에 관련 된 것으로 v1.0에서 바이트 정렬이 big-endian, 또는 little-endian으로 되어 있는지를 boolean형으로 표시하고 있다. GIOP v1.1에서는 이런 식별자를 플래그라는 8비트 옥테트로 표시하고, 최상위 비트부분에 바이트 정렬을 표시하고 있다. 그리고, 두 번째 최상위 비트에는 프래그먼트가 더 있는지 없는지를 표시하는 부분이고, 나머지는 현재



V1.0

| Magic | Version | Byte order | Msg type | Msg size |
|---|---|---|---|---|

V1.1

| Magic | Version | Flags | Msg type | Msg size |
|---|---|---|---|---|

Fig. 5 GIOP message header

reserved되어 있다. 그리고, **Msg-type**는 메시지 타입 식별을 위한 것으로 헤더의 뒷부분에 따라오는 메시지의 종류를 가리키고 있다. 마지막으로 **Msg-size**는 메시지 헤더의 뒷부분에 포함된 메시지의 크기를 옥테트 단위로 표시하고 있다.

### 3.2 메시지 구조

다음으로는 Request 메시지와 Reply 메시지 부분을 살펴보겠다. 기본적으로 이들 메시지의 처음 부분에는 GIOP 메시지 헤더가 들어가 있고, 다음으로 각 메시지의 헤더와 몸체 부분이 들어가게 된다.

Request 메시지 헤더는 클라이언트의 요구에 대한 정보를 담은 메시지로, Fig.6와 같은 형태를 가지고 있다. Fig.7에서 보이는 요구 메시지 헤더 정보를




Fig. 6 Request message

V1.0

| Service context | Request ID | Response expected | Object key | Operation | Requesting principal |
|---|---|---|---|---|---|

V1.1

| Service context | Request ID | Response expected | Reserved | Object key | Operation | Requesting principal |
|---|---|---|---|---|---|---|

Fig. 7 Request message header

살펴보면,

**서비스 컨텍스트**는 클라이언트에서 서버로 가는 ORB 서비스 데이터가 포함된 부분이고, **Request ID**는 요구 메시지를 식별하기 위한 것으로 응답 메시지와 같이 사용하게 된다. **Response-expected**는 요구 메시지가 오기를 원하는지, 원하지 않는지를 표시한다. **Reserved**는 v1.1에만 있는 것으로 0으로 셋팅되어 있고, 앞으로 사용을 위해 남겨두었다. **Object-key**는 호출할 객체를 식별한기 위한 것이다. **Operation**은 인터페이스 컨텍스트에 있는 IDL 식별자가 들어가 있는 부분으로 클라이언트에서 호출한 오퍼레이션 명이 들어가는 부분이다. **Requesting-principal**은 요구에 대한 대표자를 식별하기 위한 값이 들어간다.



Fig.8에 보이는 응답 메시지는 서버로부터의 결과를 클라이언트로 돌려줄 때 사용하는 메시지이다. Fig.9에서 응답 메시지 헤더 정보를 살펴보면, **Service context**는 요구 메시지의 것과 같고, **Request ID**는 특정 요구 메시지에 대한 응답이라는 것을 가리키고, **Reply-status**는 응답 메시지의 완료된 상태를 표시한다.


Fig. 8 Reply message

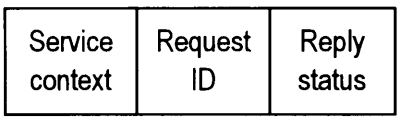



Fig. 9 Reply message header

# III. OSI 관리

망관리는 망자원의 사용과 활동을 계획, 모니터링, 요금, 조작하는 것을 말한다. 망관리의 첫 번째 목표가 모든 관리할 하위 시스템들을 한 군데의 관리자에 의해 관리하는 것이다. OSI 관리 표준 안의 목적은 일관된 방법을 제시함으로써 통합된 망관리 시스템을 정의하는 것이다(Adrian Tang,Sophia Scoggins,1994, 박희동,1991).

## 1. 관리 구조



Fig.8의 OSI 관리 모델은 관리자라고 하는 하나 또는 여러 개의 관리 프로세스와 대리자라고 불리는 프로세스들간의 상호 작용으로 기술된다.

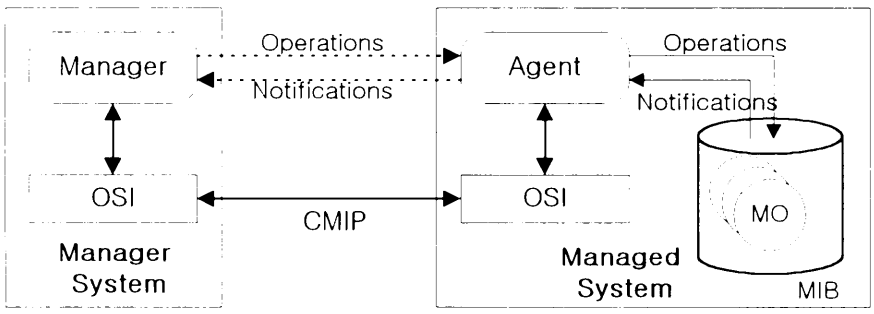



Fig. 10 Model of System Management

대리자는 관리 관점에 포함되는 관리객체들을 다루며, 관리객체들은 관리의 관점에서 모든 자원들의 추상적 표현이라 할 수 있다. 관리객체는 물리적,

추상적인 객체를 표현할 수 있고, 모든 관리객체 정보들은 MIB(Management Information Base)라는 개념적인 저장소에 저장된다. 관리 프로세스는 대리자에 의해서 유지되는 정보들을 CMIS(Common Management Information Service)에서 제공되는 서비스를 이용하여 CMIP(Common Management Information Protocol)을 통해 전달하게 된다. MIB 내에서 정보들은 관련된 속성으로 구성되며, 이들 속성들은 자체의 환경에 따라 특정 값을 가지게 된다. CMIS는 속성들의 값을 처리하는 Get, Set, Create, Delete하는 서비스를 제공하며, Event Report 서비스를 이용하는 보고 기능도 가진다. 따라서 이들 5가지 서비스들을 이용하여 피관리 객체 데이터에 대해 조작하고 보고 기능을 제공하는 것이다. 또한, CMIS에서는 피관리 객체들을 제어하기 위한 액션 서비스도 제공한다. Fig.10은 관리자와 대리자 사이의 요구와 응답에 대한 처리를 보여주고 있다.



## 2. 관리 정보 트리

관리 정보는 관리객체의 모임으로 볼 수 있는데 이들은 각각 자신의 속성을 가지고 있으며 이벤트나 액션들을 규정할 수 있는 특성이 있다. 이 관리객체의 이름은 관리 정보 트리로 구조적으로 분포되어 있고 이 관리 정보 트리는 동적으로 변경할 수 있는 특성이 있다.

관리 정보 트리의 구조는 Fig.11과 같으며 이들 중 비슷한 특성을 가진 객체들은 객체 클래스 그룹으로 분류하는 한편 이들 객체 클래스들은 다른 객체의 하위 클래스가 될 수 있는 특징을 가진다.

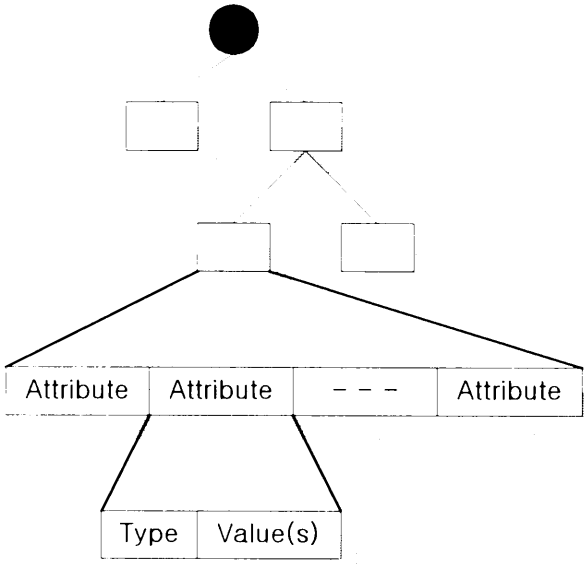



Fig. 11 Management Information Tree

# IV. 시스템 설계

CORBA에서 트래픽 모니터링하기 위한 시스템의 개발과 연구가 활발히 이루어지고 있다. 그리고, OMG에서는 CORBA와 관련해서 시스템 관리 부분의 표준화가 진행 중에 있다. 트래픽을 추출하는 방법으로 개발되어지고 있는 것들을 살펴보면, CORBA에서 응용프로그램의 작동을 살펴보기 위해 직접 ORB의 소스 코드를 변경하여 메시지가 출입할 때마다 복제하여 기록하는 방법을 사용을 하였다(Jnnalagada,1997). 그리고, ORB와 응용프로그램 사이에 메시지를 가로채는 기능을 삽입하여 메시지를 복사하는 방법과 벤더에서 제공하는 기능을 사용하여 메시지를 추출하는 방법이 있다(Inprise co.,1997, Iona co.,1997). 이는 단지 메시지를 추출하여, 이를 가지고 응용프로그램에서 CORBA상에서 트래픽 모니터링을 하려는 것이다. 그리고,



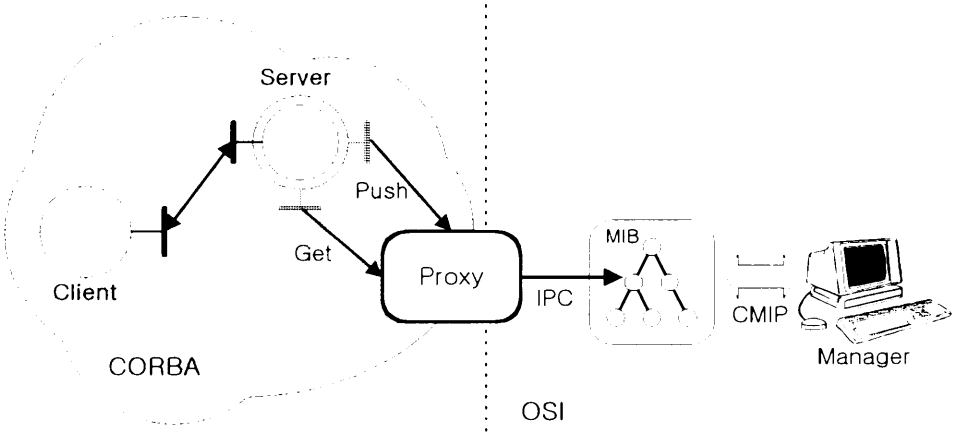



Fig. 12 Prototype of monitoring system

CORBA에서의 관리와 관련된 프로젝트로 MAScOTTE (Management Services for Object Oriented Distributed Systems)(1998)가 활발히 진행 중에 있다.

현재 구현하고자 하는 시스템의 개괄적인 흐름을 살펴보면, 다음과 같다(Fig.12). 먼저 서버 쪽에서 CORBA 객체인 프록시가 메시지에 관한 트래픽 정보를 수집을 한다. 프록시는 대리자에 있는 MIB를 갱신 하기 위한 정보를 보낸다. 그리고, 관리자가 대리자로 정보를 요구하면, 대리자는 적절한 값을 MIB에서 찾아서 관리자에게 반환 해준다.

Fig. 13에서도 보이듯이 프록시는 시스템에서 프로세스 형태로 작동하고, CORBA에서 하나의 객체로 동작을 한다. 프록시 객체의 위치는 서버 쪽이 위치하게 된다. 그리고, 프록시의 트래픽 정보를 MIB의 관리객체에 저장하여 관리자가 트래픽을 모니터링을 할 수 있게 된다. 먼저 트래픽 정보를 추출하는 부분은 아직까지 표준적인 방법이 없으므로, 특정 벤더에서



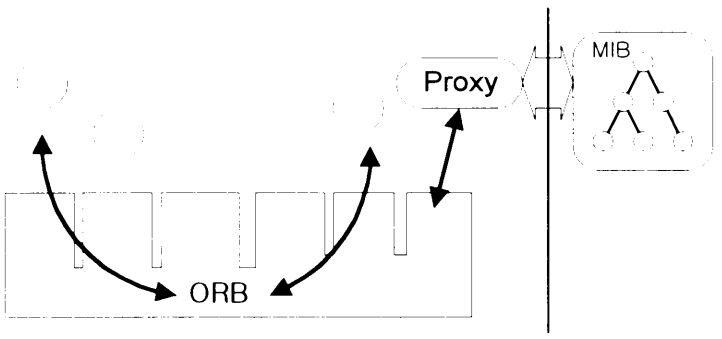



Fig. 13 The concept of monitoring system

제공되는 서비스를 사용하여 구현했다. 그리고 중간에 프록시를 두어 OSIMIS(Open System Interconnection Management Information Service)와 CORBA 서버 사이의 트래픽 정보에 대한 중계 역할을 한다.

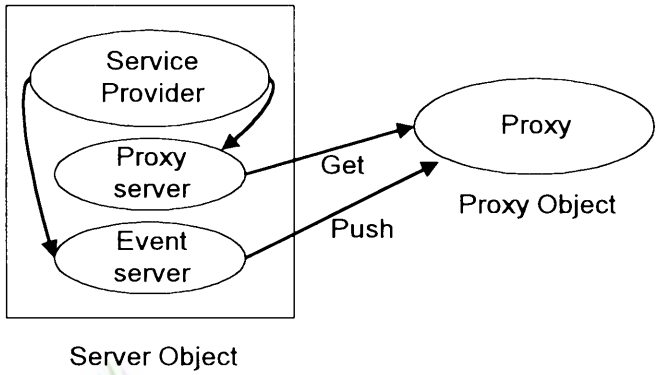



Fig. 14 Server Object Components



Fig.14은 서버 객체의 구성을 보여주고 있다. 서버 객체는 서비스 제공자와 프록시 서버, 그리고 이벤트 서버로 구성되어 진다. 일반적인 CORBA 서비스를 CORBA 클라이언트에게 제공하는 서비스 제공자와 서비스 제공자에서 트래픽 정보를 추출하여 프록시에게 트래픽 정보를 넘겨주는 역할을 하는 프록시 서버가 있다. 그리고, 서버 객체에서 발생하는 예외처리들을 이벤트 서버를 사용하여 프록시로 알려주게 되어있다.

Fig.15에서 보여주듯이 프록시 객체의 경우에는 프록시 클라이언트, 이벤트 클라이언트, 그리고 IPC(Interprocess Communication) 서버로 구성되어진다. 서버 객체로부터 트래픽 정보를 받는 프록시 클라이언트와 서버에서 예외처리 발생시 예외처리를 받는 이벤트 클라이언트가 구성된다. 마지막으로, MIB에 있는 관리객체와의 트래픽 정보를 전송하기 위해 프로세스간 통신을 담당하는

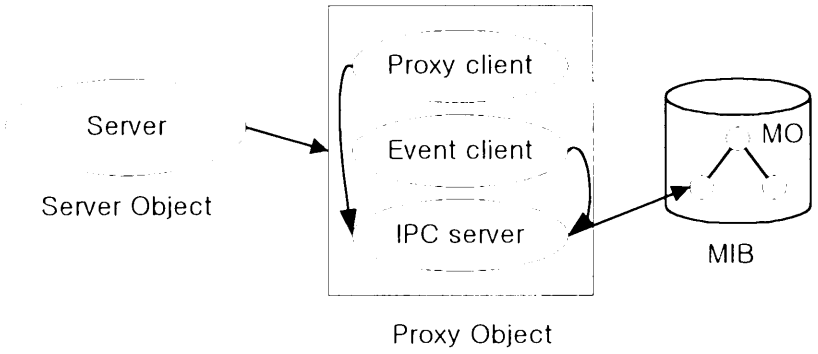



Fig. 15 Proxy Object Components

IPC 서버가 존재하게 된다.

그리고, 트래픽 파라미터들은 MIB에 있는 관리객체들로 추상화 되어 진다. 관리객체들은 하나의 객체 형태로 존재하고, 프록시 객체에 있는 IPC 서버와 프로세스간 통신을 사용하여 트래픽 정보를 얻게 된다.



본 논문에서 추출하는 트래픽 정보로는 오퍼레이션, 바인딩 시간, 예외처리, 메시지 크기, 타임 스탬프, 종료 시간이 있다. 각각의 파라미터 특징을 살펴보면, 오퍼레이션은 클라이언트가 객체 구현으로 보내는 오퍼레이션 명을 말한다. 이 정보는 요구 메시지 부분에 저장되어 있다. 이를 통하여 현재 어떤 오퍼레이션이 가장 많이 사용되는가, 즉 어떤 종류의 오퍼레이션이 빈번히 사용하는지 사용하고 있지 않은지 알 수가 있다. 바인딩 시간은 클라이언트가 요구 메시지를 보내기 위해 객체 구현과 연결 설정하는 동안의 시간을 말한다. 이를 통해서 얼마나 빠르게 연결 설정이 되는지를 알 수 있으므로, 네트워크의 부하를 알 수 있고, 로컬에서는 서버의 부하와 연결 처리 능력 등을 추측할 수 있다. 예외처리는 클라이언트와 객체구현 쪽에서

예외가 생길 경우, 어떤 예외가 생겼는지는 저장한다. 예외가 생겼다는 것은 시스템에 오류가 발생하거나, 예기치 못한 버그 등에 의한 것일 수 있으므로, 현 시스템이 얼마나 신뢰성 있게 동작하는 지를 확인 할 수 있는 파라미터이다. 메시지 크기의 현재 CORBA에서 돌아다니는 메시지 크기를 저장한다. 이 부분에서는 현재 CORBA 상에서 지나가는 메시지의 양을 가늠할 수 있어, 얼마나 많은 데이터가 지나가는지를 확인 할 수 있다. 타임 스탬프는 특정 객체 구현에서 클라이언트에서 오는 요구메시지 도착 시간을 기록 하고 있다. 그래서, 객체구현이 언제 가장 많이 사용하고 있는지를 알 수가 있다. 종료시간은 객체 구현에서 클라이언트에서 온 요구를 모든 처리를 마치고 종료하는 시간을 기록한다. 이를 통해 현재 서버가 액티브 되는 시간과 클라이언트의 요구가 얼마나 많은 처리 시간을 갖고 있는지를 계산 할 수 있다.



추출된 정보는 프록시로 모아지게 된다. 프록시는 각각의 트래픽 정보를 수집하고, MIB에 있는 관리객체의 정보들을 갱신한다. 다음으로 트래픽

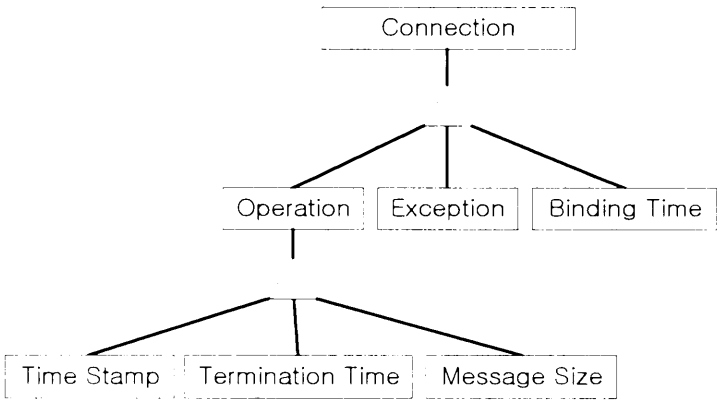



Fig. 16 Containment tree

정보를 저장할 관리객체를 정의하는 부분으로 MIB 생성 단계이다. 이 부분에서는 각각의 트래픽 파라미터들을 따로 관리객체로 정의하고, Fig. 16와 같이 맨 위에는 connection이라는 것을 두어 트리 형태로 구성했다. 대리자가 각각의 관리객체에다가 트래픽 정보를 저장하게 된다.

# V. 시스템 구현

사용한 툴로는 CORBA에는 Inprise사의 VisiBroker for C++과 CMIP은 UCL의 OSIMIS를 사용했다. 트래픽 모니터링하는 부분은 interceptor 기능을 사용했다. 그리고, 모든 트래픽 측정은 객체구현을 기준으로 작성을 하였다. 즉, 객체구현 측에 들어오는 오퍼레이션, 들어오고 나가는 메시지를 중점으로 하여 다루었다.

구현 부분은 크게 서버 객체, 프록시 객체, 그리고 관리객체 등으로 나누어 살펴 보겠다. 먼저 서버 객체를 살펴 보면, 서버 객체는 세 개의 객체로 구성되어 있다. 이들 세 개의 객체로는 서비스 제공자, 프록시 서버, 그리고 이벤트 서버가 있다.



서비스 제공자는 일반적인 서비스를 제공하는 객체로 본 논문에서는 VisiBroker에서 제공되는 기본 예제에 있는 샘플을 사용하였다. 그리고, 프록시 서버와 클라이언트의 데이터 전송을 위해 서로간의 인터페이스를 정의한 IDL 파일을 만들어서, 이를 가지고 컴파일하여 나온 소스코드를 이용하여 작성하였다. 그리고, 이벤트 서버는 CORBA에서 제공하는 이벤트 서비스를 사용하여 작성하였다. 이는 이벤트 인터페이스에 맞추어 이벤트 서버를 작성하면 된다. 서버 객체는 일반적인 CORBA 서비스 제공자에다가 프록시 서버 부분과 이벤트 서버 부분을 덧붙여진 것이다. 여기에서 프록시 서버 부분과 이벤트 서버 부분을 라이브러리 형태로 작성하였다. 이는 서버 객체 생성할 때 기존의 서비스 제공자와 같이 링크하여 만들면, 트래픽

정보를 모니터링 하는 부분이 추가되게 하였다.

시스템 구성의 두 번째 구성 요소인 프록시 객체 부분을 살펴 보자. 프록시 객체 부분도 마찬가지로 모두 세 개의 객체로 구성되어 있다. 이 들 객체로는 프록시 클라이언트, 이벤트 클라이언트, 그리고 IPC 서버 등이 있다. 먼저 프록시 클라이언트 객체는 앞의 프록시 IDL에서 생성된 소스를 가지고 구현 하였다. 이벤트 클라이언트도 마찬가지로 이벤트 서비스의 인터페이스에 맞추어 작성하였다. 다음으로 IPC 서버로 관리객체와 프로세스간 통신을 통하여 데이터 전송을 처리하고 있다. 프록시 객체는 CORBA 객체 형태로 동작하며 다른 CORBA 객체처럼 다룰 수 있다.

MIB에 있는 각 트래픽 파라미터에 대한 관리객체들은 GDMO에 의해 정의되어지고, 이를 컴파일 하여 나온 소스를 가지고 관리객체를 구현하여 사용한다. 관리객체에서 트래픽 정보를 가져오는 부분은 IPC와 프로세스 간 통신에 의해 트래픽 정보를 가져와서 값을 갱신하게 된다. 프로세스간의 통신은 메시지 큐를 사용하였다.

OSIMIS에서 관리객체을 정의하는 부분은 각각의 트래픽 파라미터를 하나의 관리객체로 정의하고 각각의 특성에 맞게 하이라킥하게 구성하였다. Fig.16는 관리객체를 구성하는 트리를 보여주고 있다.

각각의 관리객체들은 트래픽 정보가 저장된 프록시을 불러서 각 트래픽 데이터를 얻어서 관리객체에 값을 넣는다. 값이 정확히 들어갔는지 않았는지는 CMIS browser를 사용하여 확인을 하였다. Fig. 17과 Fig. 18과 같이 CMIS browser로 결과를 확인한 것이다

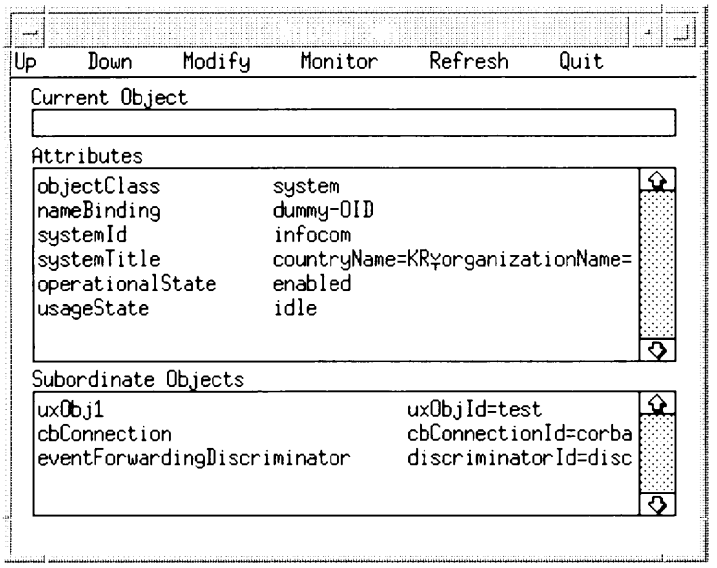



Fig. 17 Objects in the OSI agent



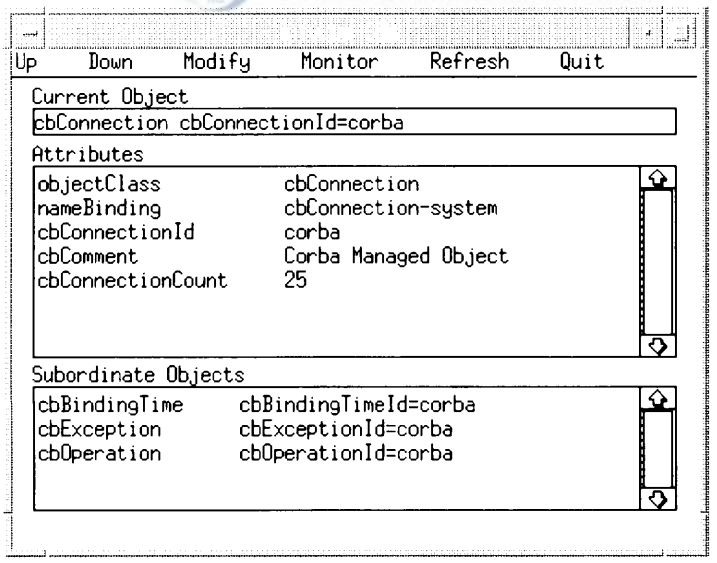



Fig. 18 Subobjects of cbConnection

# VI. 결론

CORBA는 분산 시스템에서 업계표준으로 자리잡고 있는 상황에서 많은 연구소와 대학에서 연구와 개발이 되고 있는 기술이다. 본 논문에서는 CORBA 관리에서 시스템 모니터링을 하는 것은 단지 정보수집만을 위한 것이 아니라, 더욱 효율적인 CORBA 시스템 관리를 하고자 하는데 있다. 본 논문에서는 서버 부분에 예외처리를 위한 이벤트 서버와 트래픽 정보 전달을 위한 프록시 서버부분을 구했다. 이렇게 한 것은 예외 처리가 발생할 때, 처리의 효율을 높이자는 이유가 있다. 그 외의 데이터들은 프록시 서버를 통해서 프록시 객체의 요구가 있을 경우에 전송하게 했다. 프록시 부분에서는 예외처리를 받는 이벤트 클라이언트와 트래픽 정보를 받는 프록시 클라이언트, 그리고 MIB로 트래픽 정보를 넘겨주는 IPC 서버로 구성하여 구현했다. 그리고, 각 관리객체는 트래픽 정보를 받기 위한 IPC 클라이언트로 구성이 된다. 각각의 처리들을 모듈화하여 처리함으로써 작업의 효율을 높일 수 있었고, CORBA 객체와 관리객체 객체 사이의 통신을 무리 없이 적용할 수 있었다. 각 트래픽 파라미터들은 속성에 따라 서로 간의 관계를 구성하였고, 관리객체도 마찬가지로 같은 형태로 작성하였다. 그래서, 관리자는 정보를 획득할 때 좀더 구조적으로 트래픽 데이터를 가져올 수 있게 하였다.

본 논문에서 목적은 비 표준적인 CORBA 시스템 관리를 OSI 관리자를 이용하여 CORBA의 트래픽 모니터링을 하는 것이다. 이를 위해 중간 매개자 역할을 하는 프록시를 설계했으며, 각 트래픽 정보를 CORBA에서 파라미터

형태로 정의하고, 각 파라미터는 MIB에서 관리객체형태로 정의하였다.

구현 시스템은 CORBA 트래픽에 대한 모니터링을 할 수 있는 것으로, 향후 시스템 관리 응용으로의 개발이 기대된다. 그리고, 시스템 관리에 대한 표준안이 완성이 되면 일반적인 CORBA 모니터링 시스템으로 개발이 기대된다. 그리고, 표준화된 CORBA 모니터링 시스템은 CORBA 시스템 관리에 있어서 중요한 역할을 할 것이고, 나아가 분산 환경에서 망관리에 활용되리라 사료된다.

# 참고 문헌


Adrian Tang, Sophia Scoggins, 1994, OPEN NETWORKING WITH OSI, pp.392~397.

BLACK & WHITE co., 1992, http://www.blackwhite.com/

G. Pavlou, K. MacCarthy, S. Bhatti, and G. Knight, 1995, The OSIMIS Platform: Making OSI Management Simple. Proc of the Fourth International Symposium on Integrated Network Management, pp.480~493.

H. Onazawa, 1997. 분산 오브젝트 지향 기술 CORBA, 홍릉과학 출판사, pp.11~15, pp20~29.

Inprise co. 1997, "Visibroker Manager", http://www.inprise.com/visibroker /products/vbroker /tools/.

Iona co. 1997, "Orbix Manager", White Paper.

Lakshmikanth S. Jnnalagada, 1997. "The role of network traffic statistics in devising object migration policies", pp.

MAScOTTE project, 1998, Introduction to MAScOTTE, White Paper, pp.

박희동, 1991, 관리 정보 범주 결정을 위한 OSI 관리 프로토콜의 성능 분석, pp.7~21.

OMG, 1989, http://www.omg.org/.

OMG, 1995, System management: Common Management Facilities, Volume 1, Version 2., X/Open Company.

OMG, 1998, CORBA/IIOP 2.2 Specification, pp.

Rovert Orfali, Dan Harkey, 1998, Client/Server Programming with JAVA and CORBA 2nd Edition, Joh Wieg & Sons, Inc., pp.7~17.

William Stallings, 1993. SNMP SNMPv2 and CMIP, Addison-Wesely, pp.375~381.