濟州大學校 産業技術硏究所 論文集, 第9卷, 2號, pp.142-148, 1998. 12
Cheju National Univ. R.I.I.T. Jour., Vol.9, No.2 pp.142-148, 1998. 12

CORBA 환경에서 트래픽 모니터를 위한 프록시 설계

박 재 성* · 송 왕 철**

Design of the Proxy to Monitor Traffic in CORBA environment

Jae-Seong Park* and Wang-Cheol Song**

ABSTRACT

System monitoring, a part of the system management, is a vital function for the proper operation of a system in use. So far, many companies and research laboratories have been developing various kinds of CORBA monitoring tools to obtain information about CORBA traffic. Currently, OMG is studying CORBA system management in the purpose of efficient control of CORBA system itself. In this paper, we study on the traffic monitoring for CORBA resources through OSI management. So, We has designed the proxy to monitor the CORBA traffic by OSI management method. To manage CORBA traffic resources, 6 parameters, have been made into MOs. The monitoring system consist of CORBA server, Proxy object, MIB, and Manager. CORBA server made up of Service Provider, Proxy server, and Event server. Proxy object acts as a process of CORBA object. It's made up of Proxy client, Event client, and IPC server.

Key words : CORBA, OSI, Proxy, Traffic monitoring

I. 서 론

시스템의 특성이나 정보를 얻기 위한 부분이 시스템 모니터링이다. 시스템 모니터링은 시스템의 성능을 알아보는 것 외에도 시스템과 관련된 모든 내용을 파악하여 최적의 환경을 구성하거나 관리하기 위한 정보를 얻는데 있다. 그러나, CORBA (Common Object Request Broker Architecture) 에서 시스템 관리 부분은 상당히 더딘 발걸음을 하고 있다. 현재 기본적인 표준안이 제안되어 있을 뿐이고, 벤더들이 이를 가지고 구현하기에는 부족하다. 이에 분산 환경을 위한 관리 표준안이 필요하게 되고 있고, 현재 연구가 활발히 진행중에 있다. 그러나 아직 까지는 시스템 관리 부분에 있어서 표준안이 없기 때문에 각 벤더에서 개발한 프로그램들은 자신들이 제공하는 서비스 등을 통해서 구현되고 있다. 더욱이 CORBA의 시스템 관리는 각 벤더들의 어플리케이션에 의해서만 가능하다. 본 논문에서는 이런 단점을 극복하고자 모니터링하는 것에 있어서 표준화된 방법인 CMIP (Common Management Information Protocol) 이용하여 구현을 했다. 그래서, OSI (Open Systems Interconnection) 관리에 의한

* 제주대학교 대학원
Graduate School, Cheju Nat'l Univ.
** 제주대학교 정보공학과
Dept. of Information Eng., Cheju Nat'l Univ.

CORBA 트래픽 모니터링을 하기 위한 프록시 설계를 했다.

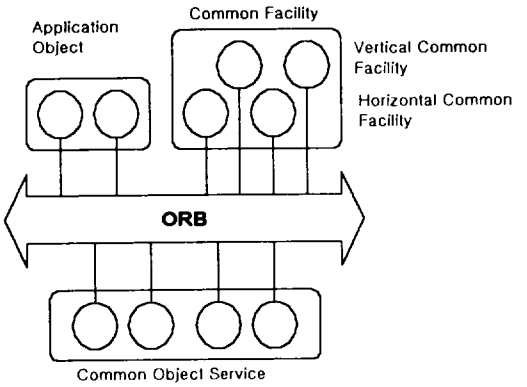

Fig. 1 Object reference model

II. CORBA

2.1. CORBA의 개요.

CORBA 환경에서는 분산 응용 프로그램의 개발이 쉽게 가능해지고, 다양한 환경과 다양한 프로그래밍 언어에 접목되어 상당히 매력적인 아키텍처이다. 더욱이 CORBA는 객체지향성 개념이 가미되어 시스템 접근에 있어서 강력하고 다양한 서비스를 제공할 수 있다. 또 한 가지 주목할 점을 CORBA는 이질적인 프로그래밍 언어들을 사용하여 개발할 수 있다는 점이 있다. 이러한 것을 가능할 수 있게 하는 기술이 IDL (Interface Definition Language)의 매핑이다.[1]

2.2. 참조 모델

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

ORB (Object Request Broker)는 객체간의 요구와 응답의 전송을 제공하는 메시징 기능을 제공한다. 공통 객체 서비스 (COS)는 객체를 구현하기 유지하기 위한 여러 가지 서비스를 제공하기 위한 집합이다. 공통 퍼실리티 (CF)는 어플리케이션에서 이용되는 기능을 제공하는 클래스 객체의 집합이다. 마지막으로, 어플리케이션 객체 (AO)는 사용자 어플리케이션 고유의 오브젝트를 말한다. 특히, ORB는 이들 객체간의 통신을 담당하는 CORBA 스펙에서 중요한 부분이다. CORBA에서는 이외에도 인터페이스 정의 언어인 IDL과 이를 다른 언어로 매핑하는 스펙, 그리고 각종 서비스들이 제공하는 구성요소와 인터페이스를 규정하고 있다.

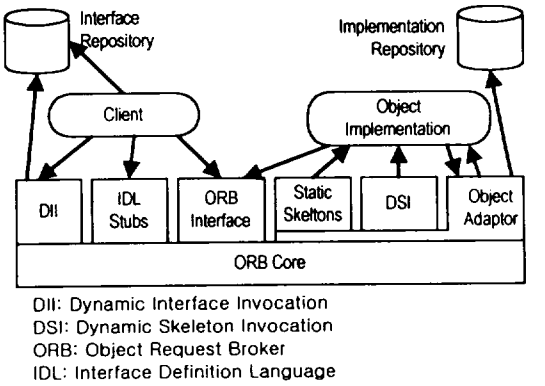

Fig. 2 CORBA frame work

2.3. ORB의 구성

ORB는 클라이언트에서 서버로의 요구를 넘겨주는 역할을 한다. 이 ORB는 Fig. 2에서 보여주듯이 ORB 코어, OA(Object Adopter), ORB 인터페이스, 동적기동 인터페이스, 통적 스켈레톤 인터페이스, IDL 스터브, IDL 스켈레톤, 인터페이스 리포지트리 및 임플리멘테이션 리포지트리로 구성된다.

클라이언트는 요구를 ORB를 통하여 객체 구현 쪽으로 발행해서, 결과를 얻어 오게 된다. 객체 구현은 서비스를 제공하는 것으로 클라이언트가 원하는 서비스를 찾아서 요구를 넘겨주게 된다. CORBA에서의 참조 정보가 있는 임플리멘테이션 리포지트리로 객체 구현에 관한 정보를 저장하고 있다. 인터페이스 리포지트리는 클라이언트 측의 인터페이스 정보를 저장하고

있다. 대 부분이 벤더의 실장에 의존하고 있다. 가운데 있는 ORB 인터페이스는 ORB에 있는 기능을 사용하거나 ORB 초기화하기 위한 인터페이스를 제공한다. 마지막으로 객체 어댑터 부분으로 객체 구현으로 메시지를 넘기는 디스패칭 기능 외에도 객체의 생성과 삭제, 활성화와 비활성화, 예외처리 등을 행한다.

2.3. GIOP

본 논문은 CORBA 가운데에서 중점적으로 다루는 부분이 GIOP (General Inter-ORB protocol) 부분이다. CORBA에서 메시지를 주고받는 타입을 정해 놓은 것이 GIOP이다. 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.[2]

본 논문에서는 메시지에서 트래픽 데이터를 추출하려고 한다. GIOP 스팩 중에서 메시지 포맷에 대해 살펴보도록 하자. 먼저 GIOP 메시지 헤더부분을 분석해 보자. Fig. 3에서 보면, 먼저 GIOP 메시지임을 표시하는 magic이라는 부분이 있다. 이 곳에는 4 바이트 크기의 GIOP라는 글자가 대문자 형태로 인코딩되어 들어가 있다. 그리고, GIOP 메시지 헤더 부분에 version부분에는 버전에 관한 정보가 들어가 있다. GIOP는 v1.1과 v1.0인 2 가지 종류를 가지고 있다. Byte order는 바이트 정렬에 관련 된 것으로 v1.0에의 바이트 정렬이 big-endian 형태인지, little-endian으로 되어 있는지를 표시하고 있다. GIOP v1.1에서는 이런 식별자를 플래그라는 8비트 옥테트 부분에 최상위 비트 부분에 표시하고 있다. 두 번째 최상위 비트에는 프래그먼트가 더 있는지 없는지를 표시하는 부분이고, 나머지는 현재 reserved되어 있다. 그리고, msg type는 메시지 타입 식별을 위한 것으로 앞에서 말한 메시지의 종류를 가리키고 있다. GIOP v1.1에서는 프래그먼트라는 메시지 타입이 추가되어 있다. 마지막으로 msg size는 메시지 헤더의 뒷부분에 포함된 메시지의 크기를 옥테트 단위로 표시하고 있다.

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

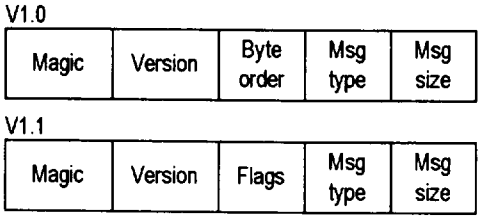

Fig. 3 GIOP message header

Fig. 4 Request message

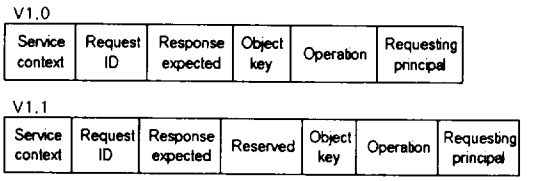

Fig. 5 Request message header

Request 메시지 헤더는 클라이언트의 요구를 담은 메시지로 Fig. 4와 같은 형태를 가지고 있다. Fig. 5에서 보여주는 헤더 정보를 살펴보면,

서비스 컨텍스트는 클라이언트에서 서버로 가는 ORB 서비스 데이터가 포함된 부분이고, Request ID는 응답 메시지와 같이 사용하게 된다. Response-expected는 요구 메시지가 오기를 원하는지, 원하지 않는지를 표시한다. Reserved는 v1.1에만 있는 것으로 0으로 세팅되어 있고, 앞으로 사용을 위해 남겨두었다. Object-key는 호출한 목적지에서 온 것인지를 식별하기 위한 것이다. Operation은 클라이언트에서 호출한 오퍼레이션이 들어가 있는 부분이다. Requesting-principal은 요구하는 당사자를 식별하기 위한 값이 들어간다.

Fig. 6에 보이는 Reply 메시지는 서버로부터의 결과를 클라이언트로 돌려줄 때 사용하는 메시지이다. Fig. 7에서 보여주는 Reply 메시지 헤더 정보를 살펴보면, Reply-status는 응답이 완료된 상태를 표시하고, 응답

메시지의 몸체 내용 부분을 결정한다. 나머지는 Reply 메시지와 같다.

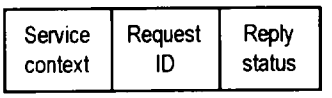

Fig. 6 Reply message header

Fig 7. Reply message

III. OSI 관리

망 관리는 망 자원의 사용과 활동을 계획, 모니터링, 요금, 조작하는 것을 말한다. 망 관리의 첫 번째 목표가 모든 관리할 하위 시스템들을 관리자에 의해 관리하는 것이다. OSI 관리 표준안의 목적은 일관된 방법을 제시함으로써 통합된 망 관리 시스템을 정의하는 것이다. [3][4]

3.1. 관리 구조

OSI 관리 모델은 관리자라고 하는 하나 또는 여러 개의 관리 프로세스와 피관리자(Agent)라고 불리는 프로세스들간의 상호 작용으로 기술된다. 피관리자는 관리 관점에 포함되는 피관리 객체(MO: Managed Object)들 다루며, 피관리 객체들은 관리의 관점에서 모든 자원들의 추상적 표현이라 할 수 있다. 피관리 객체는 물리적, 추상적인 객체를 표현할 수 있고, 모든 피관리 객체 정보들은 MIB (Management Information Base)라는 개념적인 저장소에 저장된다. 관리 프로세스

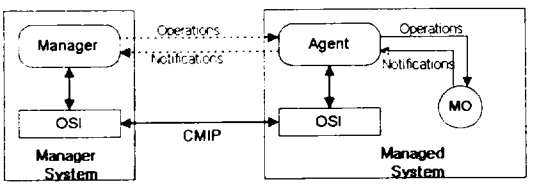

Fig. 8 Model of system management

는 피관리자에 의해서 유지되는 정보들을 CMIS (Common Management Information Service) 에서 제공되는 서비스를 이용하여 CMIP을 통해 전달하게 된다.[5]

MIB 내에서 정보들은 관련된 속성으로 구성되며, 이들 속성들은 자체의 환경에 따라 특정 값을 가지게 된다. CMIS 는 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. CMIS 에서는 피관리 객체들을 제어하기 위한 액션 서비스도 제공한다.

IV. 시스템 설계

CORBA에서 트래픽 모니터링하기 위한 시스템의 개발과 연구가 이루어지고 있다. 이러한 개발되어지고 있는 연구들을 살펴보면,

CORBA에서 어플리케이션의 작동을 살펴보기 위해 직접 ORB의 소스 코드를 변경하여 메시지가 출입할 때마다 복제하여 기록하는 방법을 사용을 했다.[6] ORB와 어플리케이션 사이에 메시지를 가로채는 기능을 삽입하여 메시지를 복사하는 방법, 벤더에서 제공하는 기능을 사용하여 메시지를 추출하는 방법이 있다.[7][8]

위의 방법은 단지 메시지를 추출하여, 이를 가지고 독립적인 프로그램에서 CORBA상에서 트래픽 모니터링을 하려는 것이다. 현재 CORBA의 연구가 활발히 진행 중에 있고, CORBA 시스템 관리 부분의 개발이 진행 중에 있다. 그리고, CORBA 관리와 관련된 프로젝트인 MAScOTTE (Management Services for Object Oriented Distributed Systems)가 활발히 진행 중에 있다.[9]

현재 구현하고자 하는 시스템의 개괄적인 흐름을 살펴보면 다음과 같다. (Fig. 9) 먼저 서버 쪽에서의 CORBA 객체인 프록시가 메시지에 관한 트래픽 정보를 수집을 한다. 프록시는 에이전트에 있는 MIB를 갱신하기 위한 정보를 보낸다. 그리고, 관리자가 에이전트로 정보를 요구하면, 에이전트는 적절한 값을 MIB에서 찾아서 리턴 해준다.

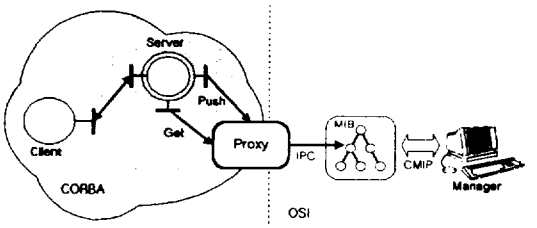

Fig. 9 The prototype of monitoring system

여기서의 프록시는 시스템에서 프로세스 형태로 작동하고, CORBA에서 하나의 객체로 작동을 한다. 그리고, 프록시 객체의 위치는 서버 쪽이 있게 된다. 프록시의 트래픽 정보를 MIB의 MO에 저장하여 트래픽을 모니터링을 할 수 있게 된다. 대략적인 개념도는 Fig. 10와 같다.

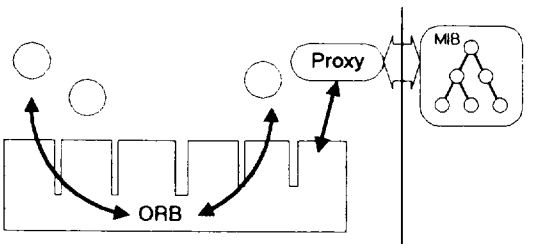

Fig. 10 Monitoring system

먼저 트래픽 정보를 추출하는 부분은 아직까지 표준적인 방법이 없으므로, 특정 벤더에서 제공되는 서비스를 사용하여 구현했다. 그리고 중간에 프록시를 두어 OSIMIS와 CORBA사이의 트래픽 정보 중계 역할을 한다. 트래픽 정보를 서버에서 추출하게 되는데, 서버 측에서는 프록시가 정보를 가져오기 위한 CORBA 객체를 인스탄스화 한다.

4.1. CORBA 모니터링 시스템

Fig. 11은 서버 객체의 인스탄스화된 모습을 보여주고 있다. 서비스를 제공하는 서버와 프록시에게 트래픽 정보를 넘겨주는 역할을 하는 프록시 서버로 구성되어 인스탄스화 하게 된다. 그리고, 프록시 서버는 서버에서 발생하는 예외 처리들을 이벤트 서버를 사용하여 프록시로 알려주게 되어 있다.

Fig. 12에서 보여주듯이 프록시의 경우에는 서버 객

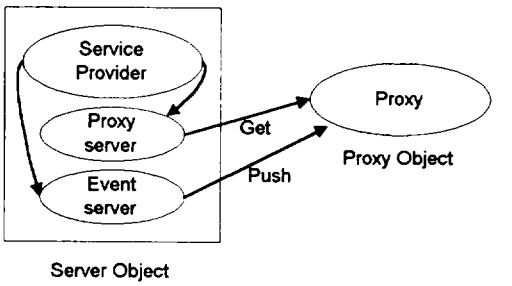

Fig. 11 Server object

체와 트래픽 정보를 주고받을 수 있는 프록시 클라이언트와 서버에서 예외처리 발생시 예외처리를 받는 이벤트 클라이언트가 구성된다. 마지막으로, MIB에 있는 MO와의 데이터 전송 위해 프로세스간 통신을 담당하는 IPC 서버가 존재하게 된다. 그리고, MO에는 프록시 객체의 IPC 서버와 통신하기 위한 IPC 클라이언트가 존재하게 된다.

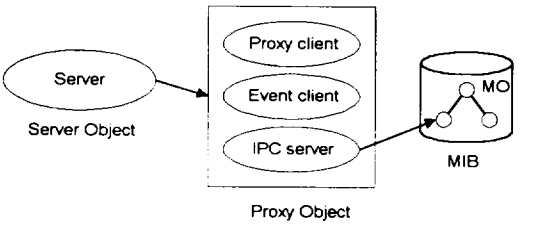

Fig. 12 Proxy object

4.2. 트래픽 정보

본 논문에서 추출하는 트래픽 정보로는 오퍼레이션, 바인딩 시간, 예외처리, 메시지 크기, 타임 스탬프, 종료 시간이 있다. 각각의 파라미터 특징을 살펴보면, 오퍼레이션은 클라이언트가 객체 구현 쪽으로 요구하는 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 추측

할 수 있다. 예외처리는 클라이언트와 객체구현 측에서 예외가 생길 경우, 어떤 예외가 생겼는지는 저장한다. 예외가 생겼다는 것은 시스템에 오류가 발생하거나, 버그 등에 의한 것일 수 있으므로, 현 시스템이 얼마나 신뢰성 있게 동작하는지를 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 기록하고 있다. 그래서, 객체구현이 언제 가장 많이 사용하고 있는지를 알 수가 있다. 종료시간은 객체 구현에서 클라이언트에서 온 요구를 모든 처리를 마치고 더 이상 할 것이 없으면 종료하는데, 바로 그 때의 시간을 기록한다. 이를 통해 현재 서버가 액티브 되는 시간과 클라이언트의 요구가 얼마나 많은 처리 시간을 갖고 있는지를 계산 가능하게 된다.

추출된 정보는 프록시로 모아지게 된다. 프록시는 각각의 트래픽 정보를 수집하고, MIB에 있는 MO에 정보들을 갱신한다. 다음으로 트래픽 정보를 저장할 MO를 정의하는 부분으로 MIB 생성 단계이다. 이 부분에서는 각각의 트래픽 파라미터들을 따로 MO로 정의하고, 맨 위에는 connection이라는 것을 두어 트리 형태로 구성했다. 에이전트가 각각의 MO에다가 트래픽 정보를 저장하게 된다. (Fig. 13)

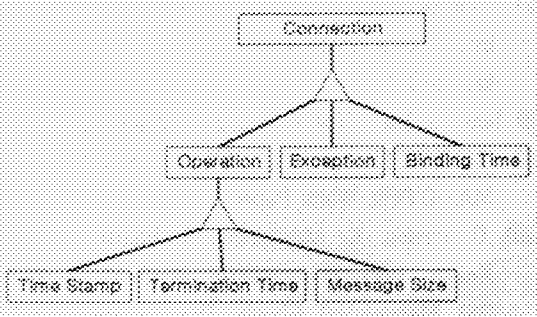

Fig. 13 Containment tree

V. 시스템 구현

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

구현 부분은 크게 서버 객체, 프록시 객체, 그리고 MO등 세 가지로 나누어 구성되었다. 먼저 서버 객체를 살펴보면, 세 개의 객체로 구성되어 있다. 이 부분이 실제 구현에 있어서는 메시지를 추출하고 프록시로 데이터를 전송하는 부분이다. 세 개의 객체로는 서비스 제공 객체, 프록시 서버, 그리고 이벤트 서버가 있다. 시스템 구성의 두 번째 구성 요소인 프록시 객체 부분을 살펴보면, 시스템에서 CORBA 객체 형태인 프로세스로 구현되었다. 프록시 객체 부분도 마찬가지로 모두 세 개의 객체로 구성되어 있다. 먼저 프록시 클라이언트 객체는 앞의 프록시 IDL에서 생성된 소스를 가지고 사용된다.

OSIMIS에서 MO(Managed Object)을 정의하는 부분은 각각의 트래픽 파라미터를 하나의 MO로 정의하고 각각의 특성에 맞게 하이라키하게 구성하였다.[10] Fig. 13는 MO를 구성하는 트리를 보여주고 있다. 각각의 MO들은 각 파라미터의 트래픽 정보가 저장될 파일들을 불러서 각 MO에 값을 넣는다. 값이 정확이 들어갔는지 않았는지는 CMIS browser를 사용하여 확인을

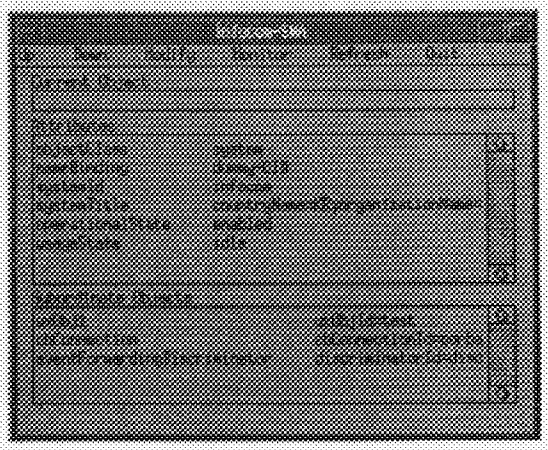

Fig. 14 Objects in the OSI agent

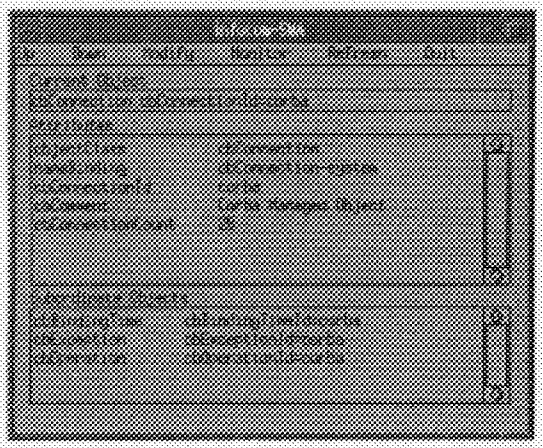

Fig. 15 Subobjects of cbConnection

여였다. Fig. 14와 Fig. 15와 CMIS browser로 확인한 것이다.

Ⅵ. 결론 및 고찰

CORBA는 분산 시스템에서 업계표준으로 자리잡고 있는 상황에서 많은 연구소와 대학에서 연구와 개발이 되고 있는 기술이다. 그러하여, 본 논문에서는 CORBA 관리에서 시스템 모니터링을 하는 것이 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 구현했다. 예외가 발생할 때, 곧바로 프록시로 전송하여 처리하게 하였고, 그 외의 데이터들은 프록시 서버를 통해서 프록시 객체의 요구가 있을 경우에 전송하게 했다. 프록시 부분의 구현에서는 예외처리를 받는 이벤트 클라이언트로 예외처리와 트래픽 정보를 받는 프록시 클라이언트, 그리고 MIB로 트래픽 정보를 넘겨주는 IPC 서버로 구성하여 구현했다. 그리고, 각 MO는 트래픽 정보를 받기 위한 IPC 클라이언트로 구성이 된다. 각각의 처리들을 객체로 처리함으로써 작업의 효율을 높일 수 있었고, CORBA 객체와 MO 객체 사이의 통신을 무리 없이 적용할 수 있었다.

본 논문에서 구현한 것은 비 표준적인 CORBA 시스템 관리를 OSI 관리자를 이용하여 CORBA의 트래픽 모니터링을 하는 것이다. 각 트래픽 정보를 CORBA에서 파라미터 형태로 정의하고, 각 파라미터는 MIB에서 MO형태로 정의하였다.

구현 시스템은 CORBA 트래픽에 대한 모니터링을 할 수 있는 것으로, 향후 시스템 관리 응용으로의 개발이 기대된다. 표준화된 CORBA 모니터링 시스템은 CORBA 시스템 관리에 있어서 중요한 역할을 할 것이고, 나아가 분산 환경에서 망 관리에 활용되리라 사료된다.

참고 문헌

1) H. Omosawa, 1997. 분산 오브젝트 지향 기술 CORBA. 홍릉과학 출판사, pp.11~25.

2) OMG, 1998. CORBA/IIOP 2.2 Specification.

3) Adrian Tang, Sophia Scoggins, 1994. OPEN NETWORKING WITH OSI, pp.332~397.

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

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

6) Lakshmikanth S. Jonalagada, 1997. The role of network traffic statistics in devising object migration policies.

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

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

9) MAS-OTTE project, 1998. Introduction to MAS-OTTE, White Paper.

10) G. Pavlou, K. MacCarthy, S. Bhatti, and G. Knight, 1996. The OSIMIS Platform: Making OSI Management Simple. Proc of the Fourth International Symposium on Integrated Network Management, pp.480~493.