# 목 차

# 그림 목차

# 표 목차

# 1. 서론

1970년대 초반, Local Area Network(LAN)의 개발과 함께 출현한 분산 시스템은 최근에 오면서 개인용 컴퓨터의 대량 보급이나 값싼 서버급 워크스테이션의 개발, 계속적인 컴퓨터 네트워크 기술의 발전, 그로 인한 데이터 전송 속도의 향상이나 데이터 전송 서비스의 질 개선에 힘입어 급격히 발전하고 다양한 분야에서 도입되어 사용되고 있다. 뿐만 아니라 분산 환경에서 제공되는 많은 서비스들이 개발되었으며 다양한 분산 기술들의 연구, 개발도 활성화되었는데 최근 가장 각광 받고 있는 분산 기술의 하나로 Common Object Request Broker Architecture (CORBA) [1]를 들 수 있다.

CORBA 는 Object Management Group (OMG)에서 제정한 표준으로 다른 종류의 (heterogeneous) 하드웨어와 소프트웨어 플랫폼 상의 분산 객체들 사이의 상호 운용성 (interoperability)을 제공하는 기술이다. 현재 몇 가지의 CORBA 제품이 개발되어 여러 분야에서 이용되고 있으며 점점 그 사용 범위가 확장되고 있는 추세이다. 현재 OMG 에 보고된 바로는 세계의 50 여 개의 회사의 프로젝트에 도입되어 이용되고 있다.

특히, 플랫폼이나 개발에 사용되는 프로그래밍 언어에 대한 독립성때문에 대규모의 분산환경에서 이미 개발된 시스템들의 통합을 위한 도구로 많이 이용되고 있다. 뿐만 아니라, 객체 모델을 바탕으로 한 기술이라는 특성으로 인해 분산 멀티미디어 시스템 개발에도 많이 이용되고 있다. Hongkong Telecom (HKT)의 경우, CORBA 와 Java 를 이용하여 개발한 다양한 Interactive Multimedia Services (IMS)를 제공하고 있다.

이와 같이 CORBA 를 이용하는 분산 서비스들이 많아지고 그것들을 이용하는 사용자들이 증가함에 따라 그러한 서비스들에 대한 관리가 절실하게 되었다. CORBA 기반의 서비스들은 대개 대규모의 복잡한 네트워크 환경 위에서 사용되며 멀티미디어 서비스와 같이 장시간의 상호작용을 필요로 하고 많은 자원을 필요로 한다는 특성이 있다. 이러한 특성들은 CORBA 기반의 서비스들이 다른 서비스들에 비해 좀 더 많은 관리 노력을 필요로 한다는 것을 말해준다.

그러나 90 년대 초반까지 네트워크 관리나 시스템 관리 분야의 활발한 연구와는 대조적으로 응용 서비스 관리에 관련된 연구는 거의 행해지지 않았다. 뿐만 아니라, 네트워크를 기반으로 하는 클라이언트-서버 모델이나 분산 환경 기반의 응용 서비스, 특히 CORBA 기반의 서비스를 위한 관리 기술의 연구는 더욱 미약했다. 하지만 네트워크의 발달과 분산 환경에서의 다양한 응용 서비스들이 각광을 받고 있는 요즈음 응용 서비스 관리의 필요성이 대두되어 연구가 활발하게 진행되고 있는 중이다 [2, 3, 4, 5].

이 논문에서는 분산 환경에서의 응용 서비스 관리에 대한 연구의 일환으로 CORBA 기반의 분산 서비스를 위한 관리 기술의 연구 결과를 제시한다. 이 연구에서는 기존의 네트워크와 시스템 관리 기술과 현재까지 진행되어 온 응용 서비스 관리 기술을 바탕으로 하며 CORBA 기반의 분산 서비스의 특성을 고려하여 CORBA 기반의 분산 서비스를 위한 관리 프레임워크를 설계하였다. 관리 프레임워크의 설계는 CORBA 기반의 분산 서비스를 위한 관리 프레임워크의 요구 사항 분석과 관리 구조 설계, 관리 서비스 정의, 그리고 관리 대상을 위해 필요한 관리 인터페이스와 관리 정보의 정의를 포함한다.

먼저 2 장에서는 관련 연구로 네트워크 및 시스템 관리의 기본 구조와 그에 해당하는 기존의 국제 표준 기술에 대해서 소개하고 응용 서비스 관리 분야에서 현재 표준화 작업이 진행되고 있는 관련 기술들에 대해서 알아볼 것이다. 3 장에서는 이 연구의 목적인 CORBA 기반의 분산 서비스 관리를 위한 요구 사항을 분석한다. 4 장에서는 3 장에서 분석한 요구 사항을 바탕으로 하여 관리 프레임워크를 설계하고 5 장에서는 4 장의 설계를 바탕으로 한 관리 시스템의 구현을 설명하고 6 장에서 이 연구를 요약하고 앞으로의 연구 방향을 제시한다.

# 2. 관련 연구

이 장에서는 CORBA 기반의 분산 서비스 관리 기술을 위한 기본 연구 단계로 조사하고 분석된 내용을 언급하겠다. 먼저 네트워크나 시스템 관리에서 기본적으로 사용되는 관리 구조를 간단하게 설명하고 기존의 관리 표준들, 즉, 인터넷 관리 구조와 OSI 관리 구조 표준에 대해서 소개할 것이다. 그리고 Internet Engineering Task Force (IETF)의 표준화 작업의 일환으로 응용 서비스 관리를 위해 정의된 여러 가지 Management Information Base (MIB)들과 CORBA 상에서의 서비스 관리에 관한 연구의 결과를 함께 소개할 것이다.

## 2.1 일반적인 네트워크 관리 구조

네트워크나 시스템 관리의 관리 구조는 일반적으로 관리자 (manager)와 대리자 (agent)로 구성된다. 관리자는 사용자로부터 관리 명령을 받아들여 대리자에게 그 명령을 수행하기 위해 필요한 요청을 하여 결과를 얻고 그것을 바탕으로 구성 관리 (configuration management), 결함 관리 (fault management), 성능 관리 (configuration management), 보안 관리 (security management), 그리고 계정 관리 (accounting management) 등 다양한 관리 기능을 수행한다. 실제로 관리되어야 할 대상들은 MO (Managed Object)로 표현되며 관리를 위해 필요한 정보들은 MIB 으로 정의된다. 대리자는 이 MIB 을 바탕으로 관리 대상들로부터 정보를 추출하여 저장하거나 관리자의 요청에 따라 적절한 정보를 제공하고 관리 대상에서 발생되는 이벤트나 결함을 감지하여 관리자에게 알려 주기도 한다.

위에서 언급된 관리 기능들은 다음과 같이 정의된다.

1. 구성 관리 기능 : 관리 대상을 정의하고 관리 대상들 간의 관계를 설정하며 그것들로부터 필요한 정보를 얻어 오거나 정보를 변화 시킴으로써 관리 대상들을 제어하는 기능이다.
2. 결함 관리 기능 : 결함 관리는 관리 대상에 결함이 발생했을 경우 이것을 빠르고 효과적으로 처리하여 시스템 및 네트워크 구성 요소들이 정상적으로 동작하도록 함으로써 안정적인 시스템 및 네트워크

환경을 제공하기 위한 기능이다.

3. 성능 관리 기능 : 성능 관리는 시스템 및 네트워크 구성 요소들의 성능을 모니터링하고 제어하여 효율적인 시스템 및 네트워크 환경을 제공하기 위한 기능이다.
4. 보안 관리 기능 : 사용이 허락되지 않은 사용자들의 시스템 및 네트워크 자원에 대한 사용을 막고 관리 정보들에 대한 접근을 제어함으로써 안전한 시스템 및 네트워크 환경을 제공하고자 하는 기능이다.
5. 계정 관리 기능 : 사용자나 사용자들의 집단에 대해 시스템 및 네트워크 자원들의 사용을 계속적으로 점검하여 자원의 사용 정도에 따라 이용료를 부과하고 사용자들이 자원을 잘못 사용하거나 비효율적으로 사용하는 문제에 대해 도움을 주기 위한 기능이다.

아래 그림 1은 위에서 설명한 일반적인 관리 구조를 보여 준다.

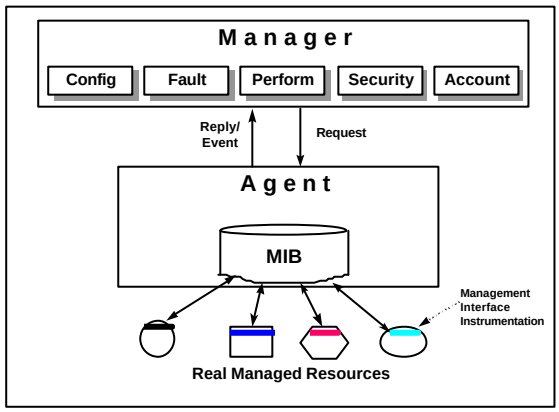


**그림 1: 일반적인 관리 구조**

## 2.2 네트워크 관리를 위한 국제 표준

이 장에서는 네트워크 관리를 위한 두 가지 국제 표준을 간단히 소개한다. 그 중 하나는 Simple Network Management Protocol (SNMP)로 대표되는 인터넷 관리 프레임워크 (Internet Management Framework)이고 다른 하나는 Common

Management Information Protocol (CMIP)으로 대표되는 OSI 관리 프레임워크 (OSI Management Framework)이다 [6]. OSI 관리 프레임워크가 네트워크 관리를 위한 궁극적인 모델이라면 인터넷 관리 프레임워크는 과도기적이고 임시적인 모델로 인식되어 있다. 하지만 인터넷 관리 프레임워크의 간결성으로 인해 현재로서는 인터넷 관리 프레임워크가 널리 이용되고 있는 상황이다.

### 2.2.1 인터넷 관리 프레임워크

인터넷 관리 프레임워크는 1990 년에 TCP/IP 를 기반으로 한 인터넷의 관리를 위해서 채택되었다 (RFC 1155, 1157, 1212, 1213) [7, 8, 9, 10][1]. 이는 TCP/IP 위에서의 프로토콜이나 서비스들도 결국은 OSI 기반의 프로토콜과 서비스들로 전환할 것이라는 예상 하에 그 전환 과정에서 필요한 몇 가지 기본적인 기능만을 제공할 수 있도록 간단하게 설계된 것이다. 1992 년에 보안 기능이 첨가된 secure SNMP (RFC 1352) [11]가 나왔고 1993 년에는 보안 기능과 그 외의 몇 가지 기능이 첨가되고 OSI 기반의 네트워크에서도 동작하는 SNMPv2 의 인터넷 드래프트 (draft)가 만들어졌다. 그리고 1996 년에는 SNMPv2 의 보안 기능의 결점을 없앤 커뮤너티 (community)기반의 SNMPv2 (RFC 1901-1908) [12, 13, 14, 15, 16, 17, 18, 19]가 나오게 되었다.

인터넷 관리 프레임워크와 관련된 표준은 크게 세가지로 나뉘어 진다. 먼저 MIB 에 포함되는 관리 정보가 어떻게 정의될 것인지를 설명하고 있는 Structure and Identification of Management Information (SMI)와 MIB, 그리고 관리자와 대리자 간의 통신 프로토콜인 SNMP 로 나누어 진다.

인터넷 관리 프레임워크의 기본 구조는 앞서 설명한 관리자-대리자 구조를 따르며 관리자와 대리자 사이의 관리 프로토콜로 SNMP 가 사용된다. SNMP 는 크게 Get, Set, Trap 의 세 가지 오퍼레이션으로 구성되어 있다.

SNMP 는 그 단순함이 CMIP 에 대한 큰 장점으로 작용하여 이미 많은 워크스테이션이나 디바이스에 SNMPv1 을 따르는 대리자가 장착되어 사용되고 있다.

---

[1] 이 프레임워크를 현재는 SNMPv1 이라 부른다.

### 2.2.2 OSI 관리 프레임워크

OSI 관리 프레임워크는 ITU-T 와 ISO 가 공동으로 제정한 것으로 X.700 시리즈로 알려져 있다. 이 프레임워크 역시 관리자-대리자 구조를 가지며 프로토콜로는 CMIP 을 사용하고 객체 지향적 방법으로 정의되는 관리 정보를 가진다.

이 표준은 크게 OSI 관리 프레임워크 개요 [20], Common Management Information Services / Common Information Management Protocol (CMIS/CMIP) [21], 시스템 관리 기능 (Systems-Management Functions) [22], 관리 정보 모델 (Management Information Model) [23, 24], 그리고 계층별 관리 (Layer Management) [25]로 나뉘어져 있다. CMIS 는 관리 정보와 명령을 교환하기 위한 응용 서비스들의 집합이고 CMIP 은 앞서 말한 것과 같이 관리자와 대리자 간에 정보와 명령들을 교환하기 위한 프로토콜이다. CMIP 에서는 event report, get, set, action, create, delete, cancel-get 의 프로토콜 오퍼레이션들이 제공된다. 그리고 두번째로 시스템 관리 기능에서는 OSI 관리 프레임워크에서 수행되어야 할 관리 기능들을 설명하고 있다. 그리고 관리 정보 모델에서는 관리 대상인 관리 객체 클래스를 정의하기 위한 내용과 구조를 설명하고 있는 Guidelines for the Definition of Managed Objects (GDMO) [24]가 포함되어 있다. 마지막으로 계층별 관리에는 OSI 레퍼런스 모델 (OSI Reference Model)의 각 계층과 관련된 관리 정보, 서비스, 기능들을 설명하고 있다. 그림 2는 OSI 시스템 관리 기능들을 보여 준다.

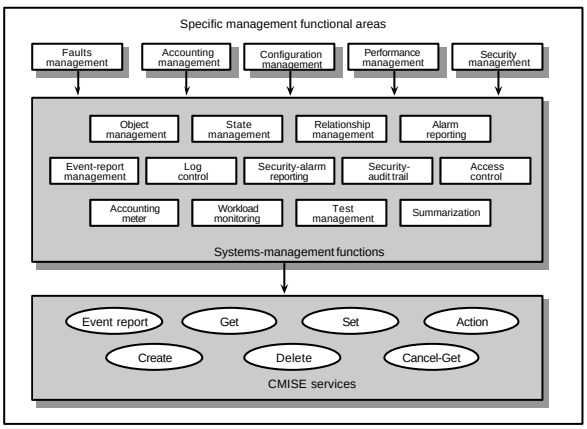


**그림 2: OSI 시스템 관리 기능**

이와 같은 OSI 의 관리 구조 및 프로토콜은 인터넷 관리 프레임워크나 SNMP 보다 많은 기능이 제공되지만 너무 복잡한 구조를 가진다는 단점이 있다. 그래서 일반적인 컴퓨터 통신망 보다는 전화망 관리에서 널리 사용되고 있다. 실제로 이 표준에 따라 구현된 시스템으로는 OSIMIS [26, 27], DSET [28], IBM TMN Family [29]등이 있다.

아래 표 1은 인터넷 관리 구조와 OSI 관리 구조를 비교한 것이다.

| 특징 | 인터넷 관리 프레임워크 | OSI 관리 프레임워크 |
|---|---|---|
| 관리 정보 모델 | 객체 기반형 (Object-based) | 객체 지향형 (Object-Oriented) |
| MIB 언어 | SNMP SMI | GDMO |
| 관리 개체간 통신 | 관리자-대리자, 관리자-관리자 | 관리자-대리자, 관리자-관리자 |
| 프로토콜 오퍼레이션 | Get, Set, Trap, 제한된 형태의 Create/Delete | M-Get, M-Set, M-Action, M-Create, M-Delete, M-Event-Report |
| 관리 객체 어드레싱 | 트리의 리프만이 정보를 가지며 OID 를 가진 관리 정보 트리 | 객체 식별자의 스코핑과 필터링 기능을 가진 관리 정보 트리 |
| 관리 응용 프로그램 | 다섯 가지의 관리 기능 | 명시되지 않음 |
| 표준화 기관 | IETF | ITU-T, ISO |

**표 1: 인터넷 관리 프레임워크와 OSI 관리 프레임워크의 비교**

## 2.3 IETF 의 응용 서비스 관리를 위한 관리 정보 표준들

이 장에서는 응용 서비스 관리 기술에 대한 관련 연구로 IETF 에서 이미 표준으로 제정했거나 표준화 작업이 진행 중인 응용 서비스 관리 관련 MIB 들을 살펴볼 것이다. 이러한 IETF 노력들은 응용 서비스 관리 기술 연구의 초기 단계의 결과로 인터넷 관리 프레임워크를 기반으로 한 관리 시스템에서 응용 서비스에 초점이 맞추어진 관리 정보를 제공할 수 있게 한다.

### 2.3.1 Host Resources MIB

Host Resources MIB (HR MIB) [30]은 호스트 컴퓨터의 자원들을 관리하기 위해 정의되었다. 이 MIB 의 내용은 호스트의 상태를 모니터링하고 호스트의 수행 능력을 짐작할 수 있게 해 준다.

이 MIB 의 내용은 크게 여섯 개의 그룹으로 구성되어 있다.

먼저 시스템 그룹으로 시스템이 가장 최근에 초기화된 때로부터의 시간, 현재 이 호스트가 정보를 저장하고 있는 사용자 세션의 수, 현재 수행 중인 프로세스의 수 등을 포함하고 있다.

두번째로는 저장 그룹으로 호스트가 가지는 저장 장소에 대한 테이블이다. 여기서 명시되는 저장 장소는 운영체제에 의해 운용되어지는 물리적인 저장 장소가 아니라 응용 프로그램에 의해 사용되는 논리적인 저장 장소이므로 테잎이나 플로피 디스크가 아니라 하드 디스크의 파티션, 파일 시스템, 램, 가상 메모리를 위한 백업 저장 장소 등에 관련된 정보를 포함한다.

세번째로는 디바이스 그룹이다. 이 그룹은 호스트에 의해 사용되는 디바이스와 관련된 정보들을 담고 있는 테이블이다. 이 테이블은 프로세서나 네트워크, 프린터, 디스크, 비디오/오디오 디바이스, 키보드, 마우스 포트, 테잎, 모니터 등과 관련된 모든 정보들을 담고 있다.

그리고 네번째는 수행 중인 소프트웨어의 그룹이다. 이 테이블은 현재 수행 중인 소프트웨어나 수행되기 위해 메모리에 로드되어 있는 프로그램에 대한 정보를 담고 있다. 여기에는 일반적인 응용 프로그램 뿐만 아니라 운영체제나 디바이스 드라이버들도 포함된다.

그리고 다섯번째로 수행 중인 소프트웨어의 성능에 대한 그룹이다. 이 그룹은 수행 중인 소프트웨어 그룹에 포함되어 있는 각 개체에 대한 CPU 사용량이나 할당 받은 메모리의 양 등의 내용을 포함한다.

그리고 마지막으로 디스크에 설치되어 있는 소프트웨어에 대한 그룹으로 호스트에 설치되어 있는 소프트웨어를 관리하고 하드웨어와 소프트웨어 간의 버전 불일치 문제 등을 찾아내는 데에 도움을 줄 수 있다.

### 2.3.2 Network Services Monitoring MIB

Network Services Monitoring MIB (NSM MIB) [31]은 네트워크 서비스를 제공하는 응용 프로그램의 관리를 위해 공통적으로 필요한 정보들을 정의하고 있다. 그러므로 NSM MIB 은 네트워크 응용 프로그램 각각의 특징적인 관리 정보를 담은 MIB 을 구성할 경우에 기본 관리 정보로서 포함될 수 있다. 대표적으로 NSM MIB을 확장하여 구성된 MIB은 뒤에 설명될 Mail Monitoring MIB과 X.500 Directory Monitoring MIB 이다. 이 세 가지 MIB 을 일컬어 Mail And Directory Management (MADMAN)이라고 한다.

NSM MIB 은 크게 두 부분으로 나뉜다. 먼저 응용 프로그램 테이블이 있다. 이것은 네트워크 서비스를 제공하는 응용 프로그램에 공통적으로 관계되는 정보를 담고 있다. 그 구체적인 내용으로는 프로그램의 이름이나 버전, 디렉토리 뿐만 아니라 최근의 초기화 시간, 수행 상태, 이 응용 프로그램으로부터 네트워크 서비스를 받고 있는 응용 프로그램, 혹은 서비스를 제공해 주는 응용 프로그램과 연결 횟수를 포함한다.

그 다음으로는 수행 중인 응용 프로그램과 다른 응용 프로그램과의 연결 상황에 대한 정보를 담고 있는 연결 테이블이다. 이 테이블에는 연결된 응용 프로그램, 사용하는 프로토콜, 응용 프로그램 타입 등이 명시된다.

### 2.3.3 Application MIB (sysAppl MIB)

Application MIB 은 응용 프로그램의 관리를 위한 MIB 이다. 이 MIB 은 다시 두 가지로 나뉘는데 하나는 시스템의 관점에서 본 응용 프로그램 관련 정보들로 응용 프로그램에 관리를 위한 변형이나 추가적인 설치 (instrumentation)가 필요하지 않는 system Application MIB (sysAppl MIB) [32]이고 또 하나는 응용 프로그램의 관점에서 본 정보들로 관리되는 응용 프로그램의 도움 없이는 추출할 수 없는 것들을 정의하고 있는 Application MIB [33]이다. 전자는 인터넷 드래프트 상태에 있으며 HR MIB 과 RDBMS MIB (Relational Database Management System MIB)에 기반을 두고 있고 후자는 IETF 의 Application Working Group (WG)에 의해 한창 정의되고 있는 중이다. 이 장에서는 전자에

대해 언급할 것이며 후자는 다음 장에서 소개된다.

응용 프로그램과 응용 프로그램을 구성하는 요소들의 관리 정보를 정의하고 있는 sysAppl MIB은 크게 두 가지 테이블로 구성된다.

먼저 그 호스트에 설치되어 있는 응용 프로그램과 그것을 구성하는 요소들에 대한 정보를 담고 있는 그룹이 있다. 응용 프로그램에 대한 정보로는 개발자, 상품 이름, 버전, 시리얼 번호, 날짜, 위치 등이 있고 구성 요소에 대한 정보로는 이 요소가 속한 응용 프로그램에 대한 인덱스, 이름, 타입, 날짜, 위치, 크기 등이 있다.

두번째 그룹은 실행 중이거나 이미 실행되었던 응용 프로그램과 그 구성 요소에 대한 정보를 담고 있다. 실행 중인 응용 프로그램에 대해서는 시작 시간과 현재 상태 등의 정보를 담고 있고 실행이 끝난 응용 프로그램에 대해서는 시작 시간, 끝난 시간, 그리고 끝날 때의 상태를 담고 있다. 그리고 응용 프로그램이 수행된 프로세스에 대한 정보들이 구성 요소 테이블에 담기게 된다.

### 2.3.4 Application MIB

Application MIB [3]은 앞서 설명한 바와 같이 sysAppl MIB의 내용을 보충하기 위해서 정의되고 있다. 이 MIB은 주로 스루풋 (throughput)이나 응답 시간 (response time)같은 성능 측정 지표들과 파일이나 I/O 디바이스와 같은 자원 관리 정보, 그리고 로깅 기능, 응용 프로그램에 대한 제어 기능에 대해 언급하고 있다.

이 MIB은 크게 6가지 그룹으로 구성되어 있다.

첫번째는 NSM MIB의 applTable과 sysAppl MIB의 sysApplRunElmtTable과의 관계를 정의하는 nsmToSysAppl 그룹이다.

두번째는 관리되는 응용 프로그램들이 사용하고 있는 파일에 대한 정보들을 담고 있는 파일 그룹이다. 파일 그룹은 사용중인 파일의 이름이나 읽기/쓰기의 요청 횟수, 읽거나 쓴 바이트의 수, 마지막으로 읽거나 쓴 시간, 파일 크기, 파일의 접근 모드 등의 정보를 담고 있다.

세번째는 커넥션 (connection) 그룹으로 수행되고 있는 응용 프로그램의 관

점에서 현재 사용중인 커넥션들에 대한 정보들을 정의하고 있다. 각 커넥션이 열린 시간, 사용되는 프로토콜, 상대편의 어드레스와 포트, 읽기/쓰기 요청 횟수, 받거나 보낸 바이트 수, 최근에 읽거나 쓴 시간 등의 정보를 담고 있다.

네번째는 ExConnection 그룹으로 예전에 사용되었던 커넥션들에 대한 정보들을 담고 있다.

다섯번째는 트랜잭션 (transaction) 그룹으로 파일이나 커넥션일 수 있는 트랜잭션 스트림의 모니터링을 위한 정보들로 트랜잭션과 관련된 요청 횟수, 요청된 데이터의 바이트 수, 최근의 요청 시간, 응답 횟수, 응답에 사용된 데이터의 바이트 수, 최근의 응답 시간 등을 포함한다.

여섯번째는 applElmtRunControl 그룹으로 수행 중인 응용 프로그램의 구성 요소들을 모니터링하고 제어하기 위한 정보들을 정의하고 있다. 힙 (heap)의 사용량이나 네트워크 커넥션의 수, 그리고 관리자가 응용 프로그램의 구성 요소를 중지시키거나 재개 시키고 재구성시키거나 끝낼 수 있도록 하는 변수들을 포함한다.

### 2.3.5 Mail Monitoring MIB

Mail Monitoring MIB (MM MIB) [34]은 전자 메일의 메세지를 전달하는 Message Transfer Agent (MTA)들을 모니터링하기 위한 MIB 이다. 이 MIB 은 NSM MIB 의 확장된 형태로 네트워크 서비스와 관련된 기본적인 관리 정보는 NSM MIB 을 이용하고 MTA 특유의 정보들을 추가적으로 정의하고 있다.

이 MM MIB 은 MTA 테이블과 MTA 그룹 테이블로 구성된다.

먼저 MTA 테이블은 각각의 MTA 에 관련된 정보들을 담고 있다. 예를 들어 받은 메세지의 수, 저장한 메세지, 전송한 메세지, 받고 저장하고 전송한 메세지의 볼륨, 메세지를 받은 수신자의 수 등을 포함한다.

두번째로 MTA 그룹 테이블은 각각의 MTA 그룹에 대한 정보를 담고 있다. 각 MTA 의 작업들을 그룹으로 나누고 그 작업에 해당하는 메세지들에 대한 정보들을 담고 있다.

### 2.3.6 X.500 Directory Monitoring MIB

500 Directory Monitoring MIB [35]은 X.500 표준안에 따른 Directory System Agent (DSA)를 모니터링하기 위한 MIB 이다. 이 MIB 또한 NSM MIB 과 HR MIB 을 확장하면서 DSA 관리를 위한 정보만을 새로이 정의하고 있다.

이 MIB 은 디렉토리 서비스에 대한 접근, 연산, 그리고 에러에 대한 통계 정보를 담고 있는 기능 테이블과 DSA 에 의해 서비스되는 엔트리들에 대한 통계 정보를 담고 있는 엔트리 테이블, 그리고 DSA 간의 통신에 대한 정보들을 담고 있는 테이블로 구성되어 있다.

먼저 기능 테이블에는 DSA 에 대한 여러 가지 형태의 바인딩과 기능, 그리고 에러 횟수가 저장된다. 그리고 엔트리 테이블에는 DSA 에 의해 서비스 직접 서비스되는 엔트리, 슬레이브, 그리고 캐쉬에 대한 통계 정보와 각 캐쉬의 성능 정보가 저장된다. 마지막으로 DSA 간의 상호 작용을 위한 테이블에는 모니터링되고 있는 DSA 와 통신하고 있는 상대편 DSA 에 대한 정보가 저장된다.

### 2.3.7 WWW MIB

WWW MIB [36]은 웹 서버를 관리하기 위한 정보를 정의하고 있는 MIB 이다. 웹 서버는 그 특성상 네트워크 서비스의 관점에서 볼 수도 있고 일반적인 응용 프로그램으로 볼 수도 있다. 뿐만 아니라 웹 서버의 관리자에 따라 여러 가지 다른 관점에서의 웹 서버의 관리 정보에 관심이 있을 수 있다. 그러므로 WWW MIB 은 NSM MIB 과 sysAppl MIB, 그리고 HR MIB 의 확장된 형태로 정의되는 것이 바람직하다.

이러한 방향으로 정의되고 있는 WWW MIB 은 시스템 그룹과 프로토콜의 사용에 대한 통계 정보 그룹, 그리고 WWW 다큐멘트에 관련된 부분이 있다.

시스템 그룹은 WWW MIB 에서 언급되는 개체들에 대한 기본적인 정보들을 나열한다. 여기서 언급되는 개체들로는 웹 서버, 웹 클라이언트, 웹 프락시 (proxy) 서버이다. 각 개체들의 어드레스, 관리인 정보, 사용하는 프로토콜 등의 정보가 포함되어 있다.

통계 정보 그룹은 각 개체 당 받은 요청/응답의 수, 전달 받은 데이터의 바이트 수 등의 정보를 포함하고 있다.

그리고 마지막으로 다큐멘트 그룹에는 다큐멘트에 처음 접근한 시간, 가장 최근에 접근한 시간, 읽힌 바이트 수, 현재 상태 등과 같은 다큐멘트에 대한 접근 통계 정보가 포함된다. 파일의 크기와 같은 다큐멘트 파일에 대한 좀 더 일반적인 정보는 sysAppl MIB 의 정보를 참조한다.

### 2.3.8 IETF 의 관리 정보 표준들의 문제점

앞에서 언급한 IETF 의 MIB 들은 CORBA 기반의 분산 서비스 관리를 위한 MIB 으로 사용하기에는 적합하지 않다. 그것은 각 MIB 들이 가진 관점이 CORBA 기반의 분산 서비스 관리의 관점과는 차이가 있기 때문이다.

Host Resource MIB 는 소프트웨어 그룹에 대한 제한적인 정보를 제공하고 있지만 주요 초점이 각 호스트의 자원에 맞추어져 있으며 Network Services Monitoring MIB 은 네트워크 서비스의 관점에서 커넥션이나 프로토콜 등의 정보가 중심이 되고 sysAppl MIB 의 경우는 단순히 응용 프로그램을 구성하는 프로세스 정보가 주를 이룬다.

Application MIB 의 경우에는 네트워크 서비스의 관점과 응용 프로그램의 자원 사용, 성능에 관한 정보를 포함하고 있지만 CORBA 기반의 서비스가 객체를 단위로 함을 고려할 때 Application MIB 의 응용 프로그램이라는 단위가 명확하지 않을 뿐더러 서비스를 구성하는 객체에 대한 정보를 얻을 수 없다.

그러므로 CORBA 기반의 서비스의 관리를 위해서는 적합한 MIB 의 정의가 우선되어야 한다. 이러한 MIB 의 정의는 MTA 를 관리하기 위한 Mail Monitoring MIB 이나 DSA 를 관리하기 위한 X.500 Directory Monitoring MIB 이나 Web 서버 등을 관리하기 위한 WWW MIB 이 추가적으로 정의되어진 것과 일맥 상통하는 일이라 할 수 있다.

J. Schönwälder 는 [37]에서 World Wide Web (WWW)의 관리를 위해 적당한 MIB 을 선택하기 위해 IETF 의 관리 정보 표준들을 비교, 분석한 결과를 제시하였다. 표 2는 WWW 를 관리하기 위한 요구 사항을 서비스 제공자의 입장에

서 정의하고 있다.

| | |
|---|---|
| 1 | CPU usage for capacity planning and fault detection |
| 2 | Disk space usage for capacity planning and fault detection |
| 3 | Network usage for capacity planning and fault detection |
| 4 | Availability of the overall server system(s) |
| 5 | Availability of the processes that make up the server(s) |
| 6 | Number of transactions handled by the server(s) |
| 7 | Number of successful and failed transactions |
| 8 | Authentication and access control monitoring |
| 9 | Error notification and error logging |
| 10 | Availability of the local IP network (percentage) |
| 11 | Connectivity to selected IP network provider |
| 12 | Network usage and network utilization |
| 13 | Response times to selected IP network provider |

**표 2: WWW 관리를 위한 서비스 제공자의 요구 사항**

다음 표 3은 위의 요구 사항을 바탕으로 IETF 의 관리 정보들을 평가한 결과 중에서 우리의 연구와 관련된 Host Resource MIB, Network Services Monitoring MIB, sysAppl MIB, Application MIB, 그리고 WWW MIB 의 결과만을 나타낸 것이다.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HR | ✍ | ✍ | | ✍ | ✍ | | | | | | | | |
| NSM | | | | | | ✍ | | | | | | | |
| SYSAPPL | | | | ✍ | ✍ | | | | | | | | |
| APPL | | | | ✍ | ✍ | ✍ | | | | | | | |
| WWW | | | | | | ✍ | ✍ | | ✍ | | | | |

**표 3: WWW 관리를 위한 요구 사항에 대한 IETF 관리 정보 표준의 평가**

표 2의 요구 사항은 단지 WWW 서비스의 관리에만 국한되는 항목은 아니다. 그러므로 위의 요구 사항을 CORBA 기반의 분산 서비스의 관리를 위한 요구 사항으로 변형시켜 이 평가에 적용시켜 보아도 표 3과 같은 결과를 얻을 수 있다. 이는 COBRA 기반의 분산 서비스 관리를 위해 새로운 MIB 이 정의되어야 한다는 것을 말해 준다. 하지만 COBRA 기반의 분산 서비스 관리를 위한 MIB 의 내용의 일부분은 Mail Monitoring MIB 이나 WWW MIB 처럼 NSM MIB 이나 sysAppl MIB 으로부터 얻어질 수 있을 것이다.

## 2.4 CORBA 기반의 응용 서비스 관리 시스템

이 장에서는 Esprit 프로젝트의 하나로 수행 중인 Management Services for Object Oriented Distributed Systems (MAScOTTE) 프로젝트 [38]에 대해 소개할 것이다. MAScOTTE 프로젝트는 1995 년 11 월에 시작한 2 년 계획의 프로젝트이다. 이 프로젝트는 CORBA 를 기반으로 하는 시스템이 증가함에 따라 이러한 시스템들의 관리 필요성이 대두되는 것에 부응하여 CORBA 기반의 시스템을 관리하고자 하는 것을 목표로 하고 있다. 프랑스의 Bull S.A., 이탈리아의 CSELT, ESA/ESRIN, 독일의 Fraunhofer-IITB, 아일랜드의 IONA, 영국의 MARI 의 6 개 회사가 공동 참여하고 있다.

### 2.4.1 MAScOTTE Project

MAScOTTE 프로젝트는 CORBA 환경에서의 관리 대상을 Object Management Architecture (OMA)의 구성요소 들로 정의하고 있다. 이 구성요소들에는 객체들과 관련된 기본 서비스인 Object Services, Object Services 보다 한 단계 위의 Common Facilities, 사용자에 의해 개발되고 사용되는 Application Objects, 객체간 통신의 중심 역할을 하는 ORB, 그리고 CORBA 객체들을 이용하는 Clients 의 다섯 가지가 포함된다.

이러한 대상들을 관리하기 위해 MAScOTTE 프로젝트에서는 크게 세 가지 부분에 대해 연구해 왔다.

첫째로 CORBA 객체들이 관리될 수 있도록 하기 위해서 기존의 객체의 인터페이스를 확장 시켜 관리 정보를 제공할 수 있도록 하는 관리 인터페이스가 있다.

두번째는 SNMP 나 CMIP 등을 기반으로 하는 외부 관리 플랫폼들과의 연동을 위한 매커니즘을 제공하는 것이다. MAScOTTE 에서 고려하는 관리 방법은 크게 두 가지로 CORBA 기반의 관리 응용 프로그램을 통하는 방법과 기존의 관리 기법을 사용한 관리 응용 프로그램, 즉 SNMP 나 CMIP 등을 사용하는 관리 프레임워크에서의 관리 응용 프로그램을 통하는 방법인데, 후자의 경우를 위해 MAScOTTE 프로젝트에서는 OSI 관리 프레임워크 기반의 응용 프로그램을 주요 대상으로 고려한 CMIP/CORBA 프락시 대리자 (proxy agent)를 제공한다. 이것은 CMIP 으로 구성된 요청을 CORBA 에서의 요청의 형태로, 그리고 CORBA 에서의 응답을 CMIP 형태의 응답 메시지로 변환시켜 주는 역할을 한다.

세번째는 관리 facilities 의 정의이다. 이 facilities 에는 크게 세가지가 있다. CORBA 의 하부구조 관리를 위한 CORBA Agent Facility 와 응용 프로그램 관점에서의 관리를 위한 Application Definition and Configuration Facility, 그리고 관리 정보를 모으고 체계적으로 관리하는 Logging Facility 와 Event Filtering Facility 가 있다. 그림 3 은 MAScOTTE 프로젝트에서 제시하는 관리 구조이다.

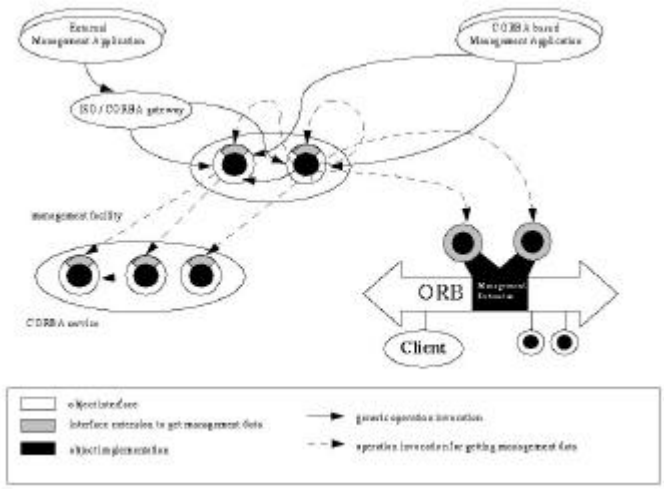


**그림 3: MAScOTTE 의 구조**

### 2.4.2 CORBA Assistant

CORBA Assistant [39]는 MAScOTTE 프로젝트의 일부로 Fraunhofer-IITB 에 의해서 진행되었다. 주요 내용은 CORBA MIB 의 정의, 관리 인터페이스 instrumentation 을 위한 라이브러리의 개발, CORBA MIB 브라우저 (orbas)의 개발, CORBA 관리 facility 일부의 개발이다. 그림 4 는 CORBA Assistant 에서 개발된 CORBA MIB 브라우저인 orbas 의 객체별 관리 정보 윈도우로 CORBA MIB 에 정의되어 있는 객체와 관련된 프로세스에 대한 정보, 객체에 대한 요청 및 데이터 관련 통계 자료를 보여 준다.

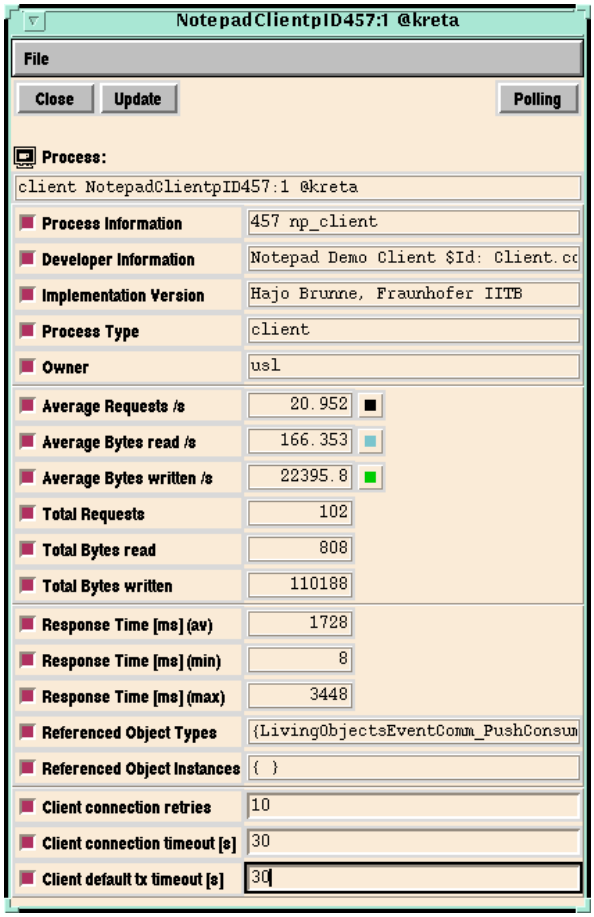


**그림 4: orbas 의 관리 정보 윈도우**

# 3. CORBA 기반의 분산 서비스 관리를 위한 요구 사항

이 장에서는 CORBA 기반의 분산 서비스 관리를 위한 요구 사항을 분석한다. 요구 사항 분석은 앞서 소개한 여러 방면에서의 관련 연구들을 비교, 평가한 결과와 관리 대상이 되는 CORBA 기반의 분산 서비스를 모델링한 결과를 바탕으로 한다.

## 3.1 CORBA 기반의 분산 서버 모델

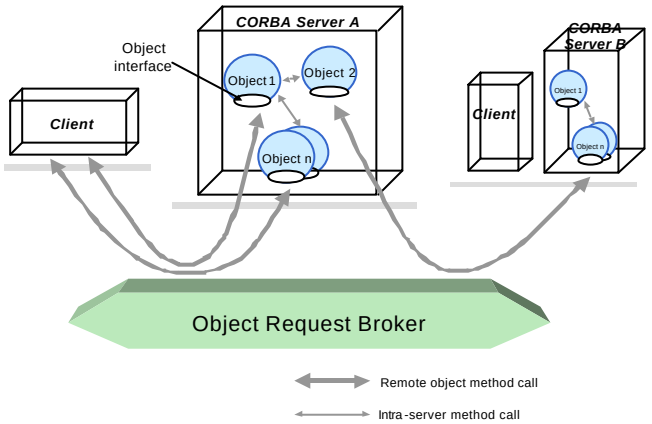


**그림 5: CORBA 기반의 분산 서버 모델**

그림 5는 CORBA 를 기반으로 한 서비스를 제공하는 서버와 그 외의 구성요소를 나타낸다. CORBA 기반의 서비스 환경을 구성하는 요소는 크게 세가지로 서비스를 제공하는 서버와 서비스를 제공 받는 클라이언트, 그리고 서버와 클라이언트 사이에서 통신 매커니즘을 제공하는 CORBA 가 있다.

CORBA 를 기반으로 하는 서비스는 원격지의 호스트에 분산되어 있는 하나 이상의 서버들로 구성된다. 그리고 각 서버들은 다시 하나 이상의 객체들을 가지는데 실제로 클라이언트와의 통신은 객체 단위로 이루어 진다. 각각의 객체는 자신이 제공하는  서비스를 위한 인터페이스를 가진다. 원격지의 클라이언트는 서버와 객체의 이름을 명시함으로써 원하는 서비스를 객체 인터페이스

를 통해 제공 받을 수 있다.

하나의 객체가 제공하는 서비스는 평범한 클라이언트 뿐만 아니라 다른 서비스 객체에 의해서도 사용될 수 있다. 이러한 경우 서비스를 제공 받는 서버 객체와 서비스를 제공하는 서버 객체 간의 관계도 평범한 클라이언트-서버 관계와 마찬가지로 취급될 수 있다.

또한, 하나의 서버 내에 존재하는 객체들간에도 상호 작용이 이루어 지는데 이러한 작용은 대부분 CORBA 상에서 객체의 인터페이스를 통해 서비스가 제공되는 형태가 아니므로 앞서 설명한 서버 객체와 클라이언트와의 관계와는 다르게 고려되어야 한다.

이 서버 모델을 바탕으로 하여 3.2 장에서는 관리 대상의 범위를 정의하고 그 대상들을 관리하기 위해 필요한 사항들을 정의할 것이다.

## 3.2 관리 시스템을 위한 요구 사항

이 연구의 목적은 CORBA 를 기반으로 하는 분산 응용 서비스의 관리이다. 이 목적에 충실하기 위해 CORBA 환경을 구성하는 여러 가지 구성 요소들, 예를 들어 Object Request Broker Core (ORB Core)나 Object Adapter, Implementation Repository 나 Interface Repository 등은 이 연구에서 관리 대상으로 고려하지 않는다. 그러므로 이 연구에서는 이를 제외한 나머지 서비스 구성 요소인 서버들과 객체들, 그리고 클라이언트 만을 관리 대상으로 고려할 것이다.

위에서 제한한 대상들을 관리하기 위해서는 우선 각각의 관리 대상에 관련된 기본 정보를 제공할 수 있어야 한다. 예를 들어 서버의 이름이나 서버가 존재하는 호스트, 서비스 제공자나 서비스를 제공하기 시작한 시간 등의 정보를 제공할 수 있어야 한다.

두번째로, 관리 대상들의 관계를 파악하고 그 관계를 제어할 수 있어야 한다. 즉, 어떤 서버에 어떤 객체가 얼마나 존재하며 얼마나 많은 클라이언트들이 서버로부터 서비스를 제공 받고 있는가 하는 정보를 제공할 수 있어야 한다. 여기서는 첫번째와 두번째의 요구 사항을 구성 관리 서비스 (configuration management service)으로 정의한다.

세번째로, 서비스 객체나 객체와 클라이언트와의 연결 등에서 어떤 일들이 이루어지고 있는지를 알 수 있어야 한다. 뿐만 아니라 서버나 클라이언트와의 연결에서 생기는 결함이나 장애가 파악되고 처리될 수 있어야 한다. 이를 결함 관리 서비스 (fault management service)이라고 한다.

네번째로, 서비스 객체들의 서비스 제공 능력, 즉 성능을 파악할 수 있어야 한다. 이는 객체에서 발생하는 장애의 원인을 찾아내는데 단서로 사용될 수 있다. 이를 성능 관리 서비스 (performance management service)이라고 한다.

마지막으로, 관리 시스템의 올바른 사용자를 확인할 수 있어야 하며 허락된 사용자에 대해서도 그 관리 범위를 제한할 수 있는 매커니즘을 제공해야 한다. 이를 보안 관리 서비스 (security management service)이라고 한다.

사실 위에서 제시한 요구 사항들 중에서 계정 관리 기능을 제외한 세 가지 기능은 서비스가 제공되는 네트워크 환경이나 시스템과 많은 관련이 있다. 그러므로 서버들의 구성이나 결함, 성능을 관리하기 위해서는 기본적으로 네트워크와 시스템 관리 기능이 제공되어야 한다. 하지만 이 연구에서의 주요 목적은 응용 서비스의 관리에 있으므로 이러한 부분의 관리는 기존의 연구 결과에 의해 충분히 제공되고 있다고 가정하기로 한다.

다음 장에서는 이러한 요구 사항을 바탕으로 CORBA 기반의 분산 서비스를 위한 관리 프레임워크을 설계하기로 한다.

# 4. CORBA 기반의 분산 서비스를 위한 관리 프레임워크의 설계

이 장에서는 3.2 장에서 정의한 요구 사항들을 바탕으로 한 CORBA 기반의 분산 서비스를 위한 관리 프레임워크의 설계에 대해 설명한다. 관리 프레임워크의 설계는 2 장에서 설명한 관련 연구들의 내용을 바탕으로 하며 지적된 문제점들을 보완하고 3.1 장에서 설명한 CORBA 기반의 분산 서비스의 특성을 반영한다.

뿐만 아니라, 2.4.1 장에서 설명된 MAScOTTE 프로젝트와는 다른 관점에서 관리 프레임워크를 설계한다. MAScOTTE 프로젝트의 연구 결과는 주로 관리 대상에 삽입되는 관리 인터페이스와 외부 관리 플랫폼과의 연동에 중점을 두는 반면 전체적으로 관리 프레임워크를 구성하는 요소들의 연관성이나 제공되는 관리 서비스의 내용에 대해서는 언급하지 않는다.

하지만 이 연구에서 수행된 관리 프레임워크의 설계는 관리 프레임워크를 구성하는 요소를 정의하고 정의된 구성 요소들로 이루어지는 전체적인 관리 프레임워크의 구조부터 설계한다. 다음으로 각 구성 요소들의 상호 동작 원리와 각각이 가지는 기능을 정의하는 것에 초점을 맞출 것이다.

먼저 4.1 장에서는 전체적인 관리 프레임워크의 구조와 구성 요소인 관리 서버, 관리 인터페이스, 관리 정보와 관리 응용 프로그램을 간단히 소개하고 그것들의 상호 관계에 대해 언급할 것이다. 그리고 4.2 4.3 4.4 장에서는 관리 인터페이스, 관리 정보, 관리 서버가 가지는 세부 기능을 구체적으로 설명할 것이다.

## 4.1 CORBA 기반의 분산 서비스를 위한 관리 구조

### 4.1.1 관리 구조 개요

이 연구에서는 분산 멀티미디어 서비스 관리 구조의 설계를 위해 객체 지향적 설계 방법론을 채택하였다. 객체 지향적 방법론은 재사용성, 추상화, 확장

성에서 우수한 방법론으로 관리 서비스를 확장하거나 관리 정보를 추가, 또는 기존의 관리 시스템과 통합하는 작업에서 유리하다. 뿐만 아니라 객체 모델을 기본으로 하는 CORBA 환경과 가장 잘 통합될 수 있는 방법론이기도 하다.

이 관리 구조는 크게 관리 서비스를 제공하는 관리 서버와 관리 대상인 분산 응용 서비스 서버, 관리 대상에 삽입되는 관리 인터페이스, 그리고 관리 응용 프로그램으로 구성된다. 이러한 구성 요소들이 이루는 전체 구조의 모습은 다음의 그림 6에서 보여진다.

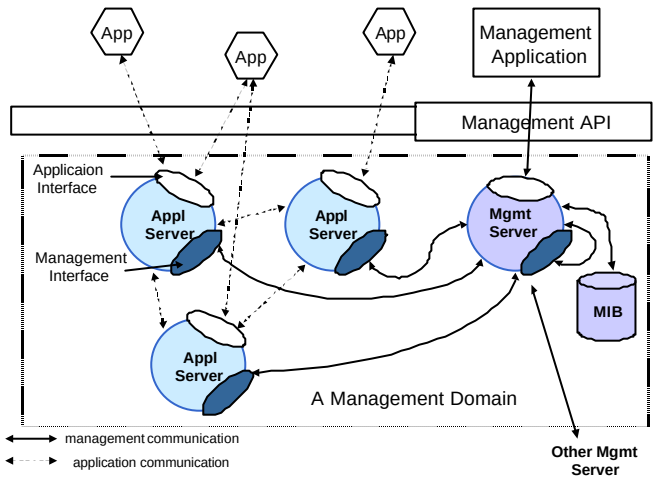


**그림 6: 관리 프레임워크의 구조**

객체 지향적 모델을 선택한 이 관리 구조는 모든 구성 요소들을 객체로 표현한다. 각각의 응용 서비스 서버들은 서버 모델에서 설명한대로 서비스를 제공하는 서비스 객체 (SO: Service Object)로 구성된다. 그리고 서버들이 관리될 수 있도록 하는 관리 인터페이스도 관리 인터페이스 객체 (MIO: Management Interface Object)로 표현된다. 뿐만 아니라 관리 서비스를 제공하는 관리 서버도 다른 응용 서버들과 마찬가지로 하나 이상의 관리 서비스 객체 (MSO: Management Service Object)로 이루어지며 관리 서비스 객체 역시 일반적인 응용 서비스 객체와 마찬가지로 관리 인터페이스 객체를 통해 스스로에 의해 관리될 수 있다. 단, 관리 응용 프로그램 (MA: Management Application)만은 객체로

표현되지 않는다.

아주 크고 복잡한 분산 환경의 경우 모든 분산 서비스를 하나의 관리 서버를 통해 관리하는 것은 힘들다. 그런 이유로 그림 6에서 보듯이 하나의 관리 서버가 관리하는 영역이 정해져 있는데 이를 관리 도메인 (Management Domain)이라고 한다. 한 도메인의 관리 서버는 다른 도메인의 관리 서버와 통신함으로써 다른 도메인의 서비스들도 간접적으로 관리할 수 있다.

### 4.1.2 관리 대상 객체와 관리 인터페이스 객체

CORBA 기반의 SO 는 서비스 제공자인 서버를 구성하는 객체로 서비스를 제공하기 위한 인터페이스를 가진다. 이것을 응용 프로그램 인터페이스 (application interface)라고 부른다. 이 인터페이스에는 서비스와 관련된 어트리뷰트 (attribute)나 오퍼레이션 (operation)들이 정의된다. 서비스의 클라이언트들은 이 어트리뷰트나 오퍼레이션을 통해 서비스를 제공 받는다.

그러나 이것들은 응용 서비스를 위한 것이므로 MSO 들이 SO 와 통신하기 위해서는 추가적인 인터페이스가 필요하다. 이것을 관리 인터페이스 (management interface)라고 한다. 각 SO 들은 객체 지향적 기법에서의 상속 기법을 이용하여 관리 인터페이스 객체 (MIO)를 상속 받음으로써 관리 인터페이스를 가지게 된다.

MIO 는 관리 정보들을 얻어내거나 수정할 수 있는 오퍼레이션을 가지고 있다. 뿐만 아니라 서버를 제어할 수 있는 오퍼레이션도 제공한다.

### 4.1.3 관리 서비스 객체

관리 서버는 몇 가지 상호 보완적인 관리 서비스들을 제공하는 객체들로 구성되는데 구성 관리 객체인 Configuration Management Service Object (cMSO), 결함 관리 객체인 Fault Management Service Object (fMSO), 성능 관리 객체인 Performance Management Service Object (cMSO), 보안 관리 객체인 Security Management Service Object (sMSO)가 있다.

MSO 들은 MA 혹은 다른 MSO 로부터 관리 서비스 요청을 받아서 처리한다. 이 요청은 하나의 MSO 내에서 처리될 수도 있지만 하나 이상의 MSO 를 필요로 하는 경우가 대부분이다. 예를 들어 fMSO 는 서버들에서 발생한 결함을 감지하기 위해 서버들의 구성 정보를 필요로 하는데 이때 cMSO 에게서 서버들의 구성 정보를 얻어낸다. 또는 MA 가 어떠한 서버에 대한 관리 정보를 cMSO 에게 요청했을 경우, cMSO 는 제일 먼저 이 요청을 한 사용자가 이 서버에 대해 권한이 있는지를 확인하기 위해 sMSO 의 서비스를 이용하기도 한다.

뿐만 아니라 MA 로부터의 요청은 MSO 를 거쳐서 어떤 SO 의 MIO 의 서비스를 필요로 하는 경우도 있다. 이런 경우에 MSO 는 MA 로부터의 요청을 적절한 SO 의 MIO 에게 전달하고 MIO 에 의해 처리된 요청의 응답은 다시 MSO 를 거쳐 MA 에게 전달된다.

그림 7은 이 관리 서비스 객체들의 세부 구조를 보여 준다.

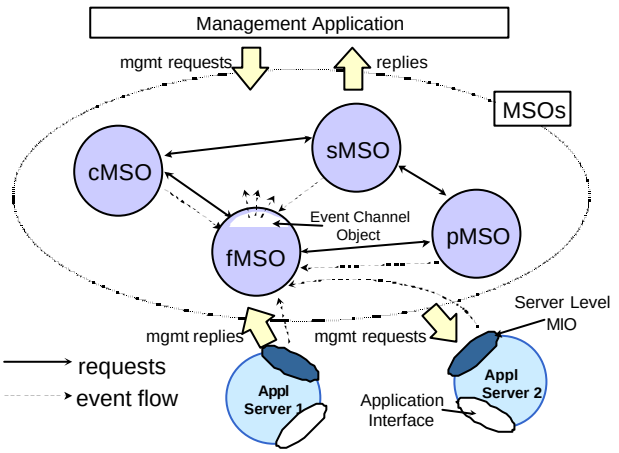


**그림 7: 관리 서비스 객체들의 세부 구조**

### 4.1.4 관리 응용 프로그램

MA 는 관리 서버에 의해 제공되는 관리 서비스를 실제 관리자가 사용할 수 있도록 해 주는 인터페이스의 역할을 한다. 실제 관리자로부터 관리 요청을

받아들이고 그 요청을 수행하기위해 관리 서버에게 여러 가지 관리 서비스들을 요청하고 그 결과를 다시 관리자에게 제공한다. MA 는 관리자가 관리 서비스를 쉽게 이용할 수 있고 관리 요청의 결과를 효과적으로 제공하기위해 주로 그래픽 인터페이스를 사용한다.

## 4.2 CORBA 기반의 분산 서비스를 위한 관리 인터페이스

4.1.2 장에서 설명한 대로 MIO 는 객체의 상속을 통해 SO 가 관리를 위한 인터페이스를 추가로 가지도록 하여 MSO 에 의해 관리될 수 있도록 한다. 이 MIO 의 역할은 크게 두 가지로 객체와 관련된 여러 가지 관리 정보를 제공하는 것과 서버의 수행을 제어할 수 있도록 하는 기능이다.

### 4.2.1 관리 인터페이스의 이중 구조

CORBA 기반의 분산 서버 모델을 고려해 볼 때, 각각의 SO 들은 독립된 객체라기보다 한 서버를 구성하는 구성 요소로서 서로 연관을 가지며 서버의 특성이나 성능을 결정짓는다고 할 수 있다. 그러므로 각 SO 들은 자신이 속한 서버를 중심으로 관리되어야 하는데 이런 성질을 만족시키기 위해 이 연구에서는 이중 구조를 가지는 관리 인터페이스를 제안한다.

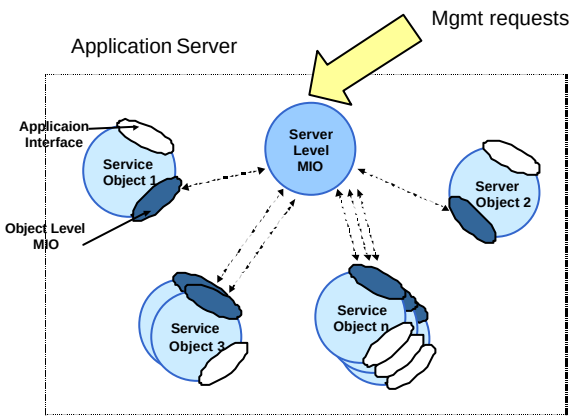

**그림 8: 이중 구조의 관리 인터페이스**

이중 구조의 인터페이스를 구성하는 요소는 두 가지이다. 하나는 서버 레벨의 MIO 이고 다른 하나는 객체 레벨의 MIO 이다.[2] 그림 8 은 이중 구조의 관리 인터페이스의 구조를 보여 준다.

### 4.2.2 서버 레벨의 관리 인터페이스 객체

서버 레벨의 MIO 는 다른 응용 서비스 객체와 마찬가지로 서버에 속하는 하나의 구성 객체로 존재한다. 한가지 다른 점이 있다면 ObjMIO 를 상속 받지 않는다는 점이다. 이 MIO 는 서버 전반에 관련된 기본 정보들을 제공하며 서버를 구성하는 객체들의 ObjMIO 들을 관리함으로써 구성 객체를 관리할 수 있는 통로를 제공한다. 뿐만 아니라 서버의 수행을 제어할 수 있는 오퍼레이션을 제공한다.

MIO 는 서버가 시작될 때 함께 시작된다. 각 서버는 초기화 단계에서 자신을 cMSO 에게 등록시킴으로써 하나의 관리 대상으로 존재하게 된다. 뿐만 아니라 MIO 인터페이스를 통해 제공되어야 할 관리 정보 중 정적인 정보들을 추출해 낸 다음 MSO 들로부터의 요청을 기다림과 동시에 자신이 속한 서버의 동작을 모니터링 한다.

MSO 로부터의 모든 요청은 MIO 를 통한다. 예를 들어, MSO 가 특정 객체에 관련된 정보를 제공 받고자 하는 경우에도 이 요청은 MIO 를 거쳐 ObjMIO 에게 전달된다. MIO 는 서버를 구성하는 모든 객체에 대한 ObjMIO 를 관리하므로 객체 관련 관리 정보의 추출도 MIO 를 통해서 이루어진다. 즉, ObjMIO 의 존재는 MSO 에게는 숨겨지는 것이다.

MIO 는 구체적으로 다음과 같은 기능을 제공한다.

✍관리 정보를 제공한다 : MIO 는 크게 두 가지 방법으로 관리 정보를 제공한다. 하나는 서버의 이름이나 소유자 같은 정적인 정보의 제공으로

[2] 이 논문에서는 서버 레벨의 MIO 는 그냥 MIO 라 부르고 객체 레벨의 MIO 는 전자와 구분하여 ObjMIO 라고 부르기로 한다.

초기화 단계에 추출해 낸 정보들을 보유하고 있다가 요청이 들어오면 즉시 제공한다. 다른 한가지는 현재의 CPU 나 메모리 사용량 등의 동적인 정보로 요청이 들어오면 그때그때 새로 정보를 추출하여 제공해 준다. MIO 가 제공하는 관리 정보들은 4.3 장에서 소개된다.

- 관리 정보를 변경시킨다 : 관리 정보의 변경이 요청되었을 경우, MIO 는 이 관리 정보가 변경 가능한 정보인지 확인한 뒤 변경 요청을 처리한다. 만약 변경이 가능하지 않은 관리 정보에 대해 요청이 들어왔다면 MIO 는 잘못된 요청이 발생했다는 경고를 fMSO 에게 전달한다.
- 서버에서 발생하는 이벤트나 장애를 fMSO 에게 보고한다 : MIO 는 계속적으로 서버를 모니터링하여 이벤트나 장애가 발생하는 경우에 fMSO 에게 보고한다. 예를 들어 사용자의 관리 정보 변경 요청이나 서버의 동작에 의해 관리 정보의 내용이 변경되었을 경우, MIO 는 이벤트를 발생 시킨다.
- 서버를 시작 또는 중지시킨다.

### 4.2.3 객체 레벨의 관리 인터페이스 객체

ObjMIO 는 각 SO 에 상속되는 객체로 SO 에 관련된 정보를 제공한다. ObjMIO 는 SO 가 생성된 후 MIO 에게 등록됨으로써 MIO 가 SO 를 관리할 수 있게 한다.

주요 기능은 MIO 와 유사하다. MIO 를 통해 전달되는 요청을 받아들여 객체 관련 정보를 제공하거나 변경시킬 수 있도록 하며 객체에서 발생되는 이벤트나 장애를 MIO 에게 보고한다. MIO 에게 보고된 이벤트나 장애는 다른 이벤트나 장애와 마찬가지로 MIO 에 의해 fMSO 에게 보고된다.

## 4.3 CORBA 기반의 분산 서비스를 위한 관리 정보

CORBA-based Distributed Services MIB (CDS MIB)은 MIO 와 ObjMIO 에 의해

추출되고 MIO를 통해 제공되는 서버와 객체 관련 관리 정보이다.

### 4.3.1 관리 정보의 분류

CDS MIB은 크게 5가지 그룹으로 나누어진다. 먼저 MIO에 의해 직접 제공되는 서버 관련 기본 정보 그룹 (general information group), 그리고 서버 전체에 대한 통계 정보 그룹 (operational statistics information group), ObjMIO에 의해 제공되는 객체 정보 그룹 (object information group), 서버에서 사용되는 여러 가지 자원들에 대한 자원 정보 그룹 (resource information group), 마지막으로 서비스 개발자에 의해 정의되는 구체 정보 그룹 (specific information group)이 있다. 구체 정보 그룹은 서버에 따라 다른 정보를 포함하고 있으므로 정해진 내용이 없다. 이 그룹은 관리 정보의 확장을 통해 채워진다.

다음 표 4에는 처음 네 가지 그룹에 속하는 정보를 구체적으로 보여 준다.

| | |
|---|---|
| 기본 정보 그룹 | 서버 이름, 호스트 이름, 실행 파일 이름, 소유자, 통신 프로토콜, 버전, 최근 수정 시간, 최근 시작 시간, 현재 상태, 플랫폼, 개발 언어 |
| 통계 정보 그룹 | 현재 인 바운드 연결 수 (current in-bound association), 받은 요청 횟수, 요청 에러 횟수, 받은 bytes, 보낸 bytes |
| 객체 정보 그룹 | 소유 객체의 수, ObjMIO 레퍼런스 테이블 |
| 자원 정보 그룹 | CPU 효율, 메모리, 자식 프로세스의 수, 쓰레드의 수, 사용중인 IPC 자원의 수 |

**표 4: CDS MIB의 정보 분류**

CDS MIB의 객체 정보 그룹에는 서버를 구성하는 객체의 ObjMIO의 레퍼런스를 관리하는 ObjMIO 레퍼런스 테이블이 있다. 이 ObjMIO 레퍼런스를 이용하여 각 객체에 대한 정보를 얻을 수 있다. 각각의 ObjMIO를 통해 제공되는 객체 관련 정보는 객체의 이름, 각 객체에 해당하는 통계 정보, 그리고 객

체의 응용 인터페이스에서 제공되는 호출 함수 (method)에 대한 정보이다.

### 4.3.2 관리 정보의 추출

표 4에 열거된 관리 정보들은 그 특성에 따라 다시 정적인 정보와 동적인 정보 두 가지로 분류된다. 동적인 관리 정보는 서버가 서비스를 제공함에 따라 그 값이 변화되는 것이고 정적인 정보는 변하지 않은 정보이다. 정적인 정보는 MIO 가 초기화될 때에 수집되어 MSO 로부터의 관리 정보 요청에 대해서 단순히 이미 수집된 정보를 제공하는 반면, 동적인 정보는 관리 정보 요청이 있을 때마다, 혹은 주기적으로 새로 수집된다.

CDS MIB 의 정보들은 크게 세 가지 소스로부터 얻어진다. 먼저 응용 프로그램으로서 혹은 프로세스로서 가지는 정보들이나 서버가 수행되는 플랫폼에 대한 정보들, 예를 들어 CPU 효율이나 자식 프로세스의 수, 혹은 운영체제의 타입은 운영체제로부터 얻어진다. 두번째로 CORBA 서버로서 가지는 정보들, 예를 들어 서버의 소유자나 인 바운드 연결 수 등은 ORB Core, Implementation Repository, Interface Repository, Object Adapter 등과 같은 OMA 구성 요소로부터 추출해 낼 수 있다. 세번째로 관리 대상인 응용 서비스 서버에 의해 직접 제공되는 정보들도 있다.

### 4.3.3 관리 정보의 확장

서버 개발자들은 구체 정보 그룹 내에 새로운 관리 정보를 정의하고 그 정보를 추출할 수 있도록 MIO 나 ObjMIO 를 확장 시킴으로써 CDS MIB 을 확장 시킬 수 있다. MIO 나 ObjMIO 의 확장은 각 관리 인터페이스 객체의 정의를 상속 받는 새로운 객체를 정의함으로써 가능하다. 특히, ObjMIO 의 경우에는 각 SO 들이 기본적으로 ObjMIO 를 상속 받기 때문에 객체 관련 정보의 추가는 더욱 용이하다.

이러한 관리 정보의 확장은 CDS MIB 이 서버에 대해 좀 더 적절한 형태를 지니도록 하여 CDS MIB 을 통한 서버의 관리가 좀 더 효과적일 수 있도록 한다.

## 4.4 CORBA 기반의 분산 서비스를 위한 관리 서비스

이 장에서는 MSO 들이 가지는 관리 서비스를 정의한다. 이 서비스들은 MSO 의 인터페이스를 통해 MA 나 MIO, 그리고 다른 MSO 에게 제공된다.

### 4.4.1 구성 관리 서비스 (Configuration Management Service)

구성 관리 서비스는 한 관리 도메인 내에 있는 서버들의 구성 정보를 관리하고 서버들을 제어할 수 있도록 하는 기능으로 cMSO 에 의해 제공된다. 한 관리 도메인 내에서 존재하는 서버들은 잠정적인 관리 대상이 된다. 이러한 잠정적인 관리 대상 중에서 관리 인터페이스를 가지며 직접 cMSO 에게 등록을 하거나 cMSO 에 의해 자동 감지된 서버들이 실제의 관리 대상이 된다.

구성 관리 서비스의 구체적인 기능은 다음과 같다.

- 관리될 서버들을 찾아내고 그 리스트를 보관한다 : cMSO 가 관리될 서버의 리스트를 보유하기 위해서는 두 가지 방법이 사용된다. 한가지는 서버가 시작될 때 직접 등록을 함으로써 자신의 존재를 알리는 방법이다. 다른 한가지는 cMSO 가 직접 관리 도메인 내에 있는 호스트들을 추적하여 관리 인터페이스를 가진 서버들을 감지해 내는 것이다.

- 서버들의 관리 상태를 알려 준다 : cMSO 가 파악하는 서버의 관리 상태는 크게 5 가지이다.

  1. 서비스는 제공하나 관리 인터페이스를 가지지 않아 실질적인 관리 오퍼레이션을 적용할 수 없는 상태
  2. 관리 인터페이스를 가지고 있고 현재 수행되지 않은 상태
  3. 관리 인터페이스를 가지고 있으며 현재 수행 중인 상태
  4. 관리 인터페이스를 가지고 있으며 현재 수행 상태가 좋지 않은 상태
  5. 관리 인터페이스를 가지고 있으나 장애로 인해 서비스를 제공할 수 없는 상태

✍서버들의 관리 정보를 변경시킨다 : cMSO 는 관리 정보 변경을 요청한 사용자가 이 서버에 대해 적절한 권한을 가지고 있는지를 확인하기 위해 sMSO 의 서비스를 받는다. sMSO 로부터 응답을 받아 권한이 있는 사용자라고 여겨질 경우에만 관리 정보를 변경 시킨다.

✍클라이언트와 서버와의 관계를 관리한다 : 서비스를 제공 받는 클라이언트들은 기본적으로 객체로 표현되지 않으므로 서버들과 같은 방법으로 관리될 수 없다. 그러므로 클라이언트와 서버와의 연결이 성립되고 해제될 때 클라이언트에 의해 생성되는 이벤트를 통해 클라이언트와 각 서버에 연결된 클라이언트들의 수와 위치를 파악하고 관리한다.

✍서버를 가동시키거나 중지시킨다 : cMSO 가 서버에서 문제점을 발견했거나 사용자로부터 요청을 받았을 경우에 서버를 재가동시키거나 중지시킨다. 서버의 재가동은 모든 서버의 동작을 처음으로 되돌리는 것이므로 서버가 가지고 있던 관리 정보들도 모두 초기화된다.

### 4.4.2 결함 관리 서비스 (Fault Management Service)

결함 관리 서비스는 서버들에서 발생되는 이벤트들을 처리하고 서버에서의 결함을 추적하여 서버들이 좀 더 안정적으로 동작할 수 있게 해 준다. 이러한 기능은 이벤트 채널과 fMSO 의 두 가지 객체에 의해 제공된다. 전자에 의한 기능을 이벤트 서비스라 하고 후자에 의한 것을 협의의 결함 관리 서비스라 한다.

#### 4.4.2.1 이벤트 서비스

이벤트 서비스는 관리 프레임워크에서 이벤트의 발생, 전달을 담당하는 기능으로 실제로는 특정 관리 기능에 국한되지 않는 서비스이다. 하지만 결함과 가장 깊은 연관을 가지므로 광의의 결함 관리 서비스에 포함시킨다. 이 관리 구조에서 이벤트의 의미는 이벤트를 받아들이는 객체로 하여금 서버들의 변화를 감지하게 하여 적절한 관리 기능을 수행할 수 있도록 해주는 촉매의 역할

이다.

이 연구에서는 이벤트 서비스를 제공하기 위한 하부 구조로 OMG 에서 정의한 CORBA 서비스 중의 하나인 이벤트 서비스 정의를 선택하였다 [40, 41]. 표준화된 CORBA 서비스 정의를 따르는 것은 이 표준을 따르는 다른 시스템 간의 호환성을 갖게 하는 장점이 있다.

이벤트 서비스의 구성 요소들은 아래와 같다. 이 구성 객체들은 OMG CORBA 의 이벤트 서비스의 명세서에 명시된 객체들과 이벤트 처리를 편리하게 하기 위해 이에 덧붙여진 이벤트 필터 객체들이 있다.

- ✍이벤트 공급 객체 (event supplier) : 이벤트 공급 객체는 이벤트를 발생시키고자 하는 어떤 구성 요소에 의해서나 소유될 수 있다.
- ✍이벤트 소비 객체 (event consumer) : 이벤트 소비 객체는 이벤트를 받고자 하는 어떤 구성 요소에 의해서나 소유될 수 있다.
- ✍이벤트 채널 (event channel) : 공급 (소비) 관리 객체를 생성한다.
- ✍공급 (소비) 관리 객체 (supplier/consumer admin) : 프락시 공급 (소비) 객체를 생성한다.
- ✍프락시 공급 (소비) 객체 (proxy supplier/consumer): 이벤트들의 멀티플렉싱 (multiplexing)을 위해 존재하는 중간자 역할의 객체이다. 하나의 프락시 소비 객체는 이벤트 채널에 존재하는 모든 프락시 공급 객체들에 연결되어 자신에게로 오는 모든 이벤트를 프락시 공급 객체들에게 제공하여 말단의 이벤트 소비 객체들에게 전달될 수 있도록 한다.

이 밖에 이벤트 필터링을 위해 이 연구에서 추가되어진 이벤트 관련 객체들은 다음과 같다. 이 객체들은 fMSO 에 의해 생성되고 관리되며 소멸된다.

- ✍이벤트 필터 객체 (event filter) : 이벤트를 필터링하기 위한 기본 객체이다. 이벤트 소비 객체를 가지며 이벤트 필터링을 위해 필요한 여러 가지 설정을 할 수 있다.
- ✍이벤트 보고 필터 객체 (event reporting filter) : 이벤트 필터 객체를 상속받는 객체로 필터링된 이벤트를 내부의 작은 크기의 큐 (queue)에 일시

적으로 저장해 둔다. 큐의 크기를 설정할 수 있는 인터페이스가 추가된다.

✍이벤트 로깅 필터 객체 (event logging filter) : 이벤트 필터 객체를 상속받는 객체로 필터링된 이벤트를 파일에 로깅하는 기능을 한다. 로깅 파일의 이름과 위치를 설정할 수 있는 인터페이스가 추가된다.

위의 객체들은 그림 9와 같은 관계로 존재하게 된다.

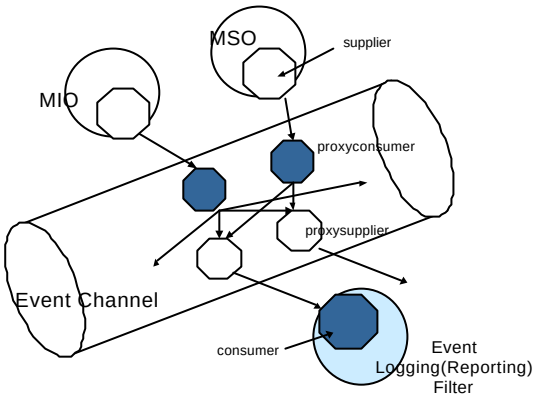


**그림 9: 이벤트 서비스 구성 객체들**

### 4.4.2.2 결함 관리 서비스

협의의 결함 관리 서비스는 여러 가지 관리 정보들과 발생된 이벤트들을 분석하여 서버들에서 생긴 장애를 감지하고 장애의 발생을 알리며 장애의 원인이 되는 결함을 찾아냄으로써 장애를 효과적으로 처리할 수 있도록 도와 주는 기능을 한다. 이 기능은 fMSO에 의해 제공된다.

결함 관리 서비스의 구체적인 기능은 다음과 같다.

✍cMSO 를 통해 각 서버의 상태 정보를 주기적으로 모니터링하여 서버에서 장애가 발생하였을 경우를 알려 준다. 즉, cMSO 가 분류하는 서버의 다섯 가지 상태 중에서 정상적인 상태에서 4 나 5 의 경우로 바뀌었을 경우, 서버에 이상이 생긴 것으로 간주한다.

✍다른 MSO 나 클라이언트로부터 발생된 이벤트들로부터 서버의 장애 여부를 판단한다.

✍장애가 발생했을 경우, 장애의 원인이 될 수 있는 결함들을 찾아낸다. 예를 들어 cMSO 에 의해 어떤 서버의 MIO 로의 요청에 실패했다는 이벤트를 받았을 경우, 그 서버가 존재하는 호스트로의 reachability 부터 ORB 의 존재 여부, 서버 프로그램의 존재 여부 등을 검사하고 가능한 원인들을 제시한다.

✍특정 경우에는 서버를 중지시키거나 초기화 시킨다.

### 4.4.3 성능 관리 서비스 (Performance Management Service)

성능 관리 서비스는 서비스 제공과 관련된 성능 지표 (performance metric) 들을 통해 서버의 성능을 모니터링하여 관리자가 서버의 성능을 보다 쉽게 관찰하고 서비스 성능 저하의 원인을 구체적으로 파악할 수 있게 해 준다. 이 기능은 pMSO 에 의해 제공된다.

성능 관리 서비스의 구체적인 기능은 다음과 같다.

✍서버의 성능에 대한 여러 가지 지표를 제공한다. : pMSO 는 서비스 처리에 대한 응답 시간, 단위 시간당 요청의 수, 단위 시간당 요청에 대한 에러 수, 단위 시간당 서버에 전달된 데이터의 양, 단위 시간당 서버에서 보내진 데이터의 양을 측정하여 제공한다. 이 값들은 ObjMIO 에 의해 누적, 기록되는 통계 정보 그룹의 데이터를 이용하여 각 객체별로 계산된다. 객체별로 계산된 값들을 이용하여 서버 전체에 대한 데이터도 함께 제공될 수 있다.

✍성능 지표에 대한 최대값, 최소값, 평균값을 제공한다.

✍과거에 측정된 성능 지표값들을 로깅하고 로깅된 결과를 제공한다.

✍원하는 서버의 성능 지표에 대해 임계값을 설정하고 지표의 값이 임계값 조건을 위배하는 경우에 보고를 받을 수 있다. 임계값 조건은 각 지표의 상한값 또는 하한값을 설정하는 것으로 이를 통해 서버에 과부하

가 생기는 것을 막을 수 있다. 응답 시간과 에러의 수에는 상한값을, 나머지 지표에 대해서는 하한값을 설정하는 것으로 한다.

관리자는 과거 성능 지표값들의 기록을 관찰함으로써 클라이언트들의 서비스 이용 실태 및 객체 각각의 특성을 파악할 수 있어 서버를 개선하거나 재설계할 때 참고할 수 있다.

### 4.4.4 보안 관리 서비스 (Security Management Service)

보안 관리 서비스는 여러 관리자가 MSO 을 이용하거나 서버들을 관리하는데 있어서의 안전을 책임진다. 대개 보안이라 하면 인증 (authentication)과 권한부여 (authorization)의 두 가지 항목으로 나누어진다. 인증은 사용자가 적법한가를 확인하는 것이고, 권한부여는 사용자가 어떤 자원에 대해 어떤 동작을 취할 때, 그 사용자가 해당 자원에 대해 해당 동작이 허용되었는가를 확인하는 것이다. 이 기능들은 sMSO 에 의해 제공되며 구체적인 기능은 다음과 같다.

- 사용자의 추가, 삭제 : sMSO 는 관리 서비스를 사용할 수 있는 사용자를 추가하거나 삭제하는 기능을 제공한다.
- 로긴, 로그아웃 : 등록되어진 사용자는 올바른 패스워드로 관리 시스템에 로긴하거나 로그아웃할 수 있다. 로긴할 때 sMSO 는 현재 사용하려고 하는 사용자가 적법한지를 확인하고, 토큰 (token)이라는 유일한 번호를 주어 사용자가 관리 서비스를 사용하려고 할 때 제시하게 함으로써 적법하지 않은 사용자에 의해 관리 서비스가 이용되는 것을 막는다.
- 자원에 대한 접근 권한 (ACL: access control list) 설정 : 서버에 대한 사용자의 권한을 설정할 수 있다. 사용자는 각 서버에 대해 1) 관리 정보를 읽을 수 있는 권한, 2) 관리 정보를 변경시킬 수 있는 권한, 3) 제어할 수 있는 권한을 가질 수 있다. 이러한 권한들을 관리하기 위해 sMSO 는 사용자의 이름과 서버의 이름과 권한의 조합으로 이루어지는 ACL 을 위한 데이터베이스를 관리한다.

# 5. CORBA 기반의 분산 서비스를 위한 관리 시스템의 구현

이 장에서는 4 장에서 설명된 관리 프레임워크의 설계를 검증하기 위해 구현된 프로토타입 시스템에 대해 간단히 소개할 것이다. CorbaMan 이라고 불리는 이 시스템은 포항공과대학교의 분산처리환경 연구실에서 개발된 CORBA 기반의 분산 멀티미디어 시스템인 MAESTRO [42]를 대상으로 하였다.

먼저 5.1 장에서는 관리 대상 시스템인 MAESTRO 에 대해 간단히 소개하고 5.2 장에서는 CorbaMan 의 프로토타입 구조와 환경, 그리고 관리 서버와 관리 인터페이스 관리 인터페이스의 구현에 대해 간단히 설명한다. 그리고 5.3 장에서는 구현된 관리 응용 프로그램을 이용하여 MAESTRO 의 서비스 관리 방법을 사용자 인터페이스 화면과 간단한 예를 이용하여 설명할 것이다.

## 5.1 MAESTRO 개요

MAESTRO 는 분산 멀티미디어 응용 프로그램들을 보다 쉽게 개발, 운용, 관리할 수 있도록 다양한 멀티미디어 서비스를 제공하는 시스템이다.

MAESTRO 에는 분산 멀티미디어 서비스를 제공해 주는 CORBA 기반의 분산 서버들이 존재한다. 이 서버들이 제공하는 서비스에는 네임 서비스 (Name Service), 통신 서비스 (Communication Service), 세션 서비스 (Session Service), 저장/인출 서비스 (Storage/Retrieval Service)가 있다. 이 서비스들을 제공하는 각각의 서버들도 역시 CORBA 객체들로 구성된다.

뿐만 아니라, 멀티미디어 응용 프로그램에서 공통적으로 필요한 멀티미디어 데이터 처리와 멀티미디어 관련 장치의 조작을 위한 멀티미디어 API 를 제공함으로써 MAESTRO 를 이용한 멀티미디어 응용 프로그램의 개발을 더욱 손쉽게 한다.

MAESTRO 의 네 가지 서비스에 대한 기능 정의는 다음과 같다.

1. 네임 서비스 : 네임 서비스는 MAESTRO 상에서 수행되는 분산 객체들에 대한 정보를 관리하고 고유한 이름을 통해 특정 객체를 찾을 수 있는 기능을 제공한다.

2. 통신 서비스 : 통신 서비스는 멀티미디어 응용 프로그램 사이에서 다양한 미디어의 데이터를 전송할 수 있는 기능을 제공한다. 특히, 비디오나 오디오 같은 연속적인 데이터를 위한 동기화 (synchronization)이나 다자간 통신 (Multi-point Communication), 흐름 제어 (Flow Control) 등의 기능을 제공한다.
3. 세션 서비스 : 세션 서비스는 공동 작업을 하고자 하는 사람들을 하나의 그룹으로 묶어 주는 서비스이다. 즉, 세션 서비스를 이용해서 하나의 세션을 생성하거나 원하는 세션을 찾거나 세션에 참여하거나 참여한 세션을 떠나거나 생성한 세션을 없앨 수 있다.
4. 저장/인출 서비스 : 저장/인출 서비스는 기본적으로 멀티미디어 API 층에서 정의된 여러 미디어 객체들을 저장하거나 저장되어 있는 미디어 객체들을 인출할 수 있도록 해 준다.

## 5.2 CorbaMan의 구현

이 장에서는 앞에서 소개된 MAESTRO 의 서버들을 관리하기 위해 구현된 CorbaMan 에 대해 설명한다. 그림 10은 CorbaMan 의 대략적인 구조를 보여 준다.

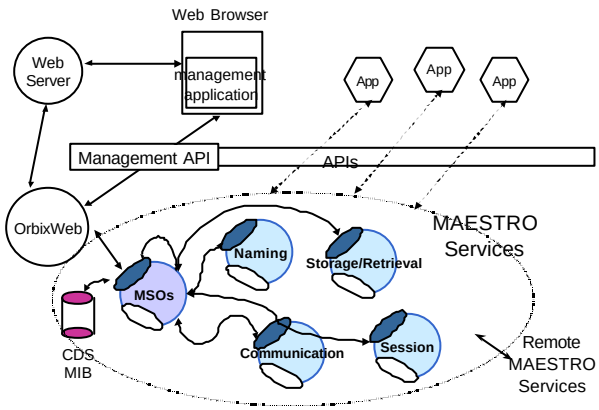


그림 10: CorbaMan의 구조

### 5.2.1 개발 환경

CorbaMan 은 IONA 에서 개발한 ORB 인 OrbixWeb 2.0.1 [43]상에서 개발되었다. CorbaMan 의 관리 인터페이스 및 관리 서버들은 C++를 사용하여 구현되었으며 관리 응용 프로그램은 Java 를 이용하여 웹 기반의 인터페이스를 가지도록 개발되었다.

Java ORB 인 OrbixWeb 은 Internet Inter-ORB Protocol (IIOP)를 이용하여 C++로 구현된 관리 서버와 Java 로 구현된 관리 응용 프로그램이 통신할 수 있도록 해 준다.

### 5.2.2 관리 Instrumentation

MAESTRO 의 서버들은 CORBA 기반의 분산 서버 모델에서 설명한 대로 여러 가지 서비스 객체로 구성된다. 이러한 서비스 객체들은 다음의 과정을 거쳐 관리 가능한 객체가 된다.

1. CORBA IDL 로 정의되는 서비스 객체의 인터페이스가 ObjMIO 인터페이스를 상속 받도록 한다.
2. 서비스 객체의 구현 클래스가 ObjMIO 의 구현 클래스를 상속 받도록 한다.
3. 서버가 시작할 때 MIO 객체를 생성시키며 생성된 MIO 객체의 레퍼런스를 cMSO 의 register_server 호출 함수를 이용하여 cMSO 에게 서버를 등록시키는 코드를 삽입한다.
4. 서비스 객체의 오브젝 코드와 MIO 관련 라이브러리들을 링크 시킨다.

필요하다면 2 와 3 사이에서 ObjMIO 의 함수들을 오버라이딩 (overriding)하여 변경, 확장 시키거나 추가적으로 필요한 정보들을 정의하고 추가 관리 정보들의 추출을 위해 ObjMIO 의 get_specific 함수를 구현함으로써 ObjMIO 를 확장시킬 수 있다.

아래는 서버 레벨 MIO 를 생성하고 cMSO 의 register_server 함수를 호출하

는 부분의 예제 코드를 보여 준다.

```
// 서버 레벨 MIO 를 생성시킨다.
MIO_i *mio_ptr = new MIO_i(server_name, domain_name, version);
// MIO 객체에 대한 레퍼런스를 스트링으로 변환시킨다.
char *str;
str=mio_ptr->_object_to_string(CORBA::IT_INTEROPERABLE_OR_KIND);
// MIO 객체 레퍼런스를 등록시키기 위해 cMSO 에 바인드한다.
cMSO_var cmso_var = cMSO::_bind(":cMSO", domain_name);
// cMSO 에 MIO 를 등록시키고 Server ID 를 부여받는다.
SID id = cmso_var->register_so(server_name, str);
```

### 5.2.3 관리 서버의 구현

관리 서비스는 구성 관리 서버, 결함 관리 서버, 이벤트 채널, 성능 관리 서버, 보안 관리 서버로 구현되었다. CORBA IDL 로 정의된 각 서버의 인터페이스는 부록 1 에 실려 있다.

### 5.2.4 관리 응용 프로그램의 구현

관리 응용 프로그램은 Java Applet 으로 구현되어 따로 설치될 필요가 없으며 어떤 플랫폼에서든지 웹 브라우저를 이용하여 실행시킬 수 있다. 뿐만 아니라, CorbaMan 의 관리 응용 프로그램은 웹을 기반으로 한 일관성 있고 쉬운 사용자 인터페이스를 제공한다.

## 5.3 CorbaMan 을 이용한 MAESTRO 의 관리 시나리오

이 장에서는 CorbaMan 을 이용한 MAESTRO 의 관리 시나리오를 몇 가지 소개한다.

### 5.3.1 구성 관리 시나리오

CorbaMan 이 실행되면 가장 먼저 해야 할 일은 적법한 사용자의 이름과 패스워드로 관리 시스템에 로긴하는 일이다. 로긴 과정을 거치면 CorbaMan 은 현재의 관리 도메인에 포함된 호스트들의 아이콘들을 보여 준다. 호스트 아이콘을 누르면 각 호스트에 존재하는 서버들을 보여 준다. 다시 원하는 서버의 아이콘을 누르면 서버에 생성되어 있는 객체들을 보여 준다. 그림 11은 이 과정에서 보여지는 관리 응용 프로그램 화면이다. 서버와 객체 아이콘은 서버의 상태에 따라 다른 색으로 나타난다.

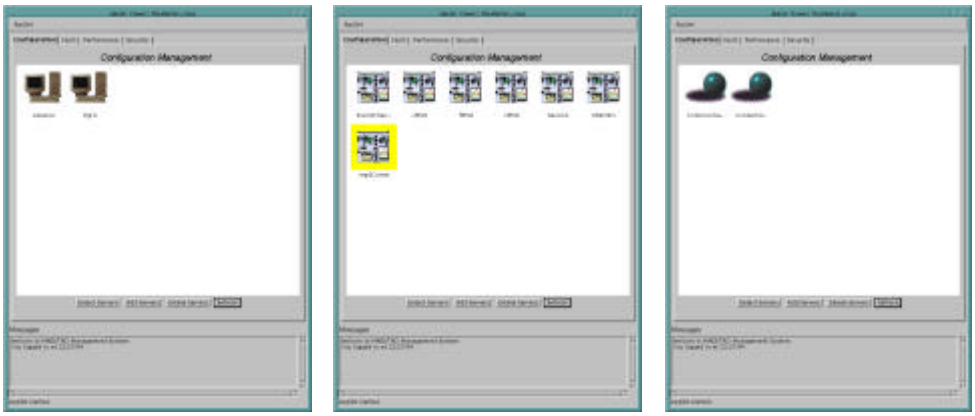

**그림 11: 구성 관리 화면**

관리 정보의 추출이나 변경은 각 서버나 객체의 아이콘의 팝업 메뉴의 항목을 이용하여 쉽게 실행될 수 있다. 단, MIO 를 가진 서버나 객체 (노란색)에 대해서만 수행 가능하다.

### 5.3.2 이벤트 필터링 및 결함 처리 시나리오

예를 들어, 구성 관리 화면을 통해 서버들의 관리 정보를 살펴보다가 한 서버의 상태가 변화하였다고 하자. 아이콘은 서버에 장애가 생겨 서비스를 제공할 수 없는 상태임을 나타내고 있다 (검은색). 이런 경우, 관리자는 결함 관리 화면을 참조함으로써 서비스 장애의 원인에 대한 정보를 얻을 수 있다.

이 경우, 서버의 상태 변화를 모니터링하고 있던 fMSO 는 서비스 수행이 중단된 서버에 대해 장애의 원인을 밝히기 위한 여러 가지 오퍼레이션을 취한다. 그 결과, 서버가 돌아가던 호스트의 ORB 역할의 프로세스가 수행을 중지했음을 발견하고 그 사실을 이벤트로 발생시켜 CorbaMan 의 모든 구성 요소들에게 알린다. 이 이벤트를 받은 관리 응용 프로그램은 이 이벤트의 내용을 메시지 영역을 통해 알려 준다. 이 메시지를 발견한 관리자는 결함이 발생한 서버와 ORB에 대해 적절한 조치를 취할 수 있다.

다른 예로, MAESTRO의 개발자가 MAESTRO의 통신 서버의 이용 상황을 파악하기 위해 이 서버에 대한 이벤트 필터를 생성시킬 수도 있다. 그림 12의 필터 생성 인터페이스를 이용하여 필터를 생성시키게 되면 아래에 필터 아이콘이 추가된다. 이 아이콘의 팝업 메뉴를 이용하여 필터링된 결과를 볼 수 있다. 그림 13은 통신 서버에 대해 이벤트 필터링을 한 결과이다.

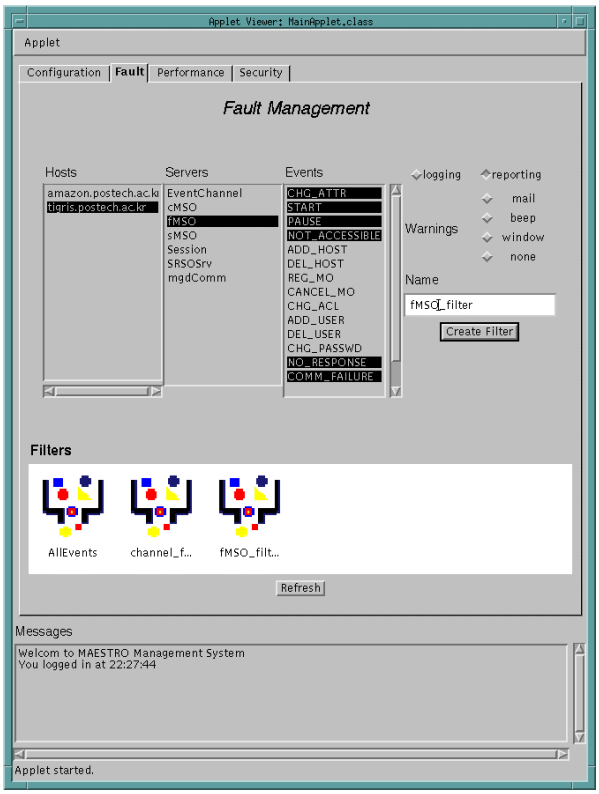


**그림 12: 결함 관리 화면**

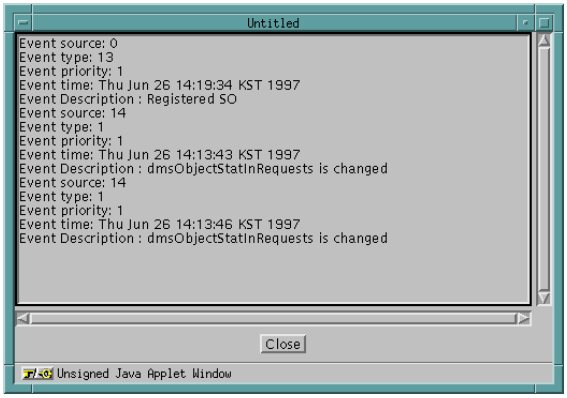


**그림 13: 통신 서버에 대한 이벤트 필터링 결과**

### 5.3.3 서버의 성능 평가 시나리오

MAESTRO 의 통신 서버는 대용량의 멀티미디어 데이터를 복잡한 과정을 거쳐 처리하는 일을 하므로 서버의 수행 능력이 MAESTRO 를 사용하는 클라이언트에게 아주 중요한 요소이다. 그러므로 MAESTRO 의 통신 서버 관리자는 성능 관리 서비스를 이용하여 MAESTRO 의 성능을 측정하고 평가하여 최적의 상태에서 서비스를 제공하기를 원할 것이다. 이 경우에 관리자는 CorbaMan 의 성능 관리 기능을 이용하여 통신 서버의 성능을 모니터링하고 성능이 악화되는 경우에 이벤트 보고를 받을 수 있다.

성능 모니터링은 성능 관리 화면에서 원하는 서버의 객체에 대해 성능 모니터링 설정을 함으로써 가능하다. 또한, 중요하게 생각되는 성능 지표에 대해 임계값을 설정하여 그 순간에 이벤트를 보고 받을 수 있다. 예를 들어, 통신 서버의 포트 객체의 단위 시간당 받은 바이트 수와 보낸 바이트 수에 대한 임계값을 설정한다. 그리고 결함 관리 서비스를 이용하여 통신 서버에서 발생되는 이벤트들만을 필터링하는 필터를 생성시키고 생성되는 이벤트들을 관찰한다. 관리자는 통신 서버가 처리하는 바이트 수가 임계값보다 적어지는 순간에 이벤트를 받게 되고 그 순간에 구성 관리 서비스를 통해 통신 서버에 존재하는 포트 객체의 수나 연결된 클라이언트의 수 등을 참고함으로써 통신 서버의 성능 과 이 수치들과의 관계를 파악하고 성능 향상을 꾀할 수 있다.

그림 14는 ‘amazon.postech.ac.kr’라는 호스트에서 수행되는 mgdComm 이라는 통신 서비스 객체에 대한 성능 지표 측정의 결과를 보여 준다.

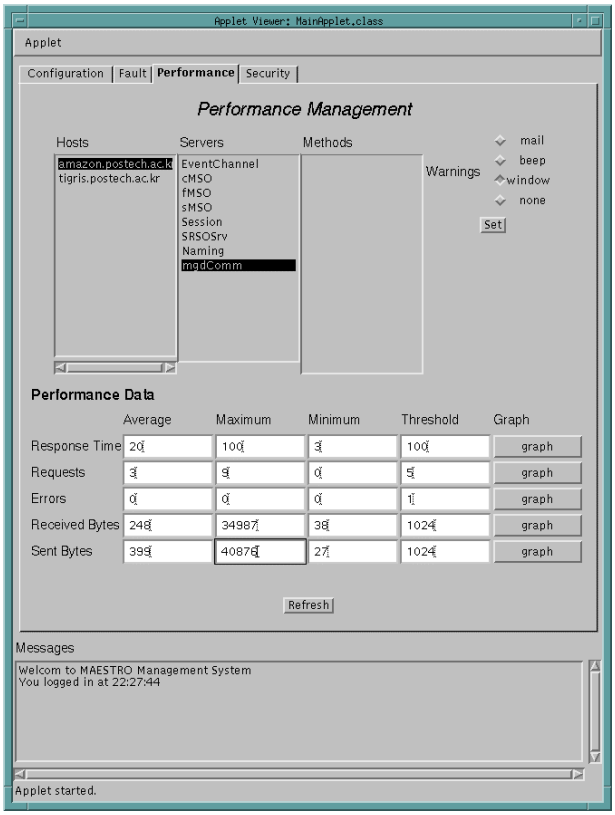


**그림 14: 성능 지표 측정 결과**

# 6. 결론 및 향후 과제

이 논문에서는 응용 서비스 관리 기술 연구의 결과로 현재까지 국내외에서 진행된 여러 가지 관련 연구들의 상황을 살펴 보았다. 뿐만 아니라 이러한 관련 연구들을 바탕으로 CORBA 기반의 분산 서비스를 관리하기 위한 관리 구조, 관리 인터페이스와 관리 정보, 그리고 관리 서비스를 정의하고 설계하였다.

CORBA 기반의 분산 서비스 관리를 위한 이 연구는 CORBA 기반의 분산 서비스를 주 관리 대상으로 삼고 있지만 특별히 CORBA 만의 특성이나 환경이 설계에 영향을 주지는 않았다. 그러므로 이 연구의 결과가 CORBA 기반의 시스템 뿐만 아니라 객체 지향적으로 설계된 다양한 분산 응용 프로그램들을 관리하는 기술 연구에도 효과적으로 적용할 수 있을 것이라고 생각한다.

뿐만 아니라 현재 OMG에서 추진되고 있는 CORBA Common Facility의 정의나 그 밖의 응용 서비스 관련 표준안 제정 작업에서도 고려될 수 있을 것이며 요즈음 주목을 받고 있는 웹 기반의 관리 시스템의 개발 사례로도 언급될 수 있을 것이다.

마지막으로 이 연구에 추가되어야 할 부분으로는 크게 두가지가 있다.

첫번째로는 MAScOTTE 에서 언급하고 있는 OMA 구성 요소에 대한 관리 기능을 제시하는 것이다. ORB나 Object Adapter, Repository 등과 CORBA Object Services 나 Common Facilities 에 대한 관리는 CORBA 기반의 서비스들의 수행에 많은 영향을 미친다. 그러므로 이러한 OMA 구성 요소에 대한 관리는 응용 서비스 관리를 좀 더 구체적이고 다양하게 할 수 있도록 도움을 줄 것이다.

두번째로는 네트워크와 시스템 관리 부분과의 연동 문제이다. 앞서 3 장에서 언급한 것처럼 이 연구에서의 응용 서비스 관리는 응용 서비스가 제공되는 시스템이나 사용되는 네트워크 환경에 대한 관리가 이미 전제되어 있는 상황에서 이루어졌다. 하지만 실제 상황에서는 그렇지 않으므로 이 연구에서 제시된 관리 구조 및 기능들과 기존의 관리 시스템과의 연동 문제는 중요한 과제 중에 하나일 것이다.

이러한 연동 문제를 해결하는 한 가지 방법으로 SNMP 나 CMIP 기반의

대리자 (agent)들과 CORBA 기반의 관리자를 연동시켜 주는 게이트웨이를 사용하는 방법이 있다.

현재 이 연구의 연장으로 SNMP-CORBA 게이트웨이 [44, 45]와 CMIP/CMIS-CORBA 게이트웨이 [45]를 개발하여 SNMP 대리자와 CMIP 대리자들에 대한 네트워크 관리 기능을 추가할 계획이다.

# 참고 문헌


[1] OMG. The Common Object Request Broker: Architecture and Specification Revision 2.0. OMG, July 1995, OMG TC Document.

[2] M. Bauer, N. Coburn, D. Erickson, P. Finnigan, J. Hong, P. Larson, J. Slonim, D. Taylor, and T. Teorey, "A distributed system architecture for a distributed application environment", *IBM Systems Journal*, 33(3): pp. 399-425, September 1994.

[3] M. Bauer, P. Finnigan, J. Hong, J. Rolia, T. Teorey, and G. Winters, "Reference architecture for distributed systems management", *IBM Systems Journal*, 33(3): pp. 426-444, September 1994.

[4] J. W. Hong, M. J. Katchabaw, M. A. Bauer, and H. Lutfiyya. "Modeling and management of distributed applications and services using the OSI management framework", *Proc. Of the International Conference on Computer Communication*, pp. 215-220, Seoul, Korea, July 1995.

[5] J. W. Hong, G. Gee, and M. A. Bauer, "Towards automating instrumentation of systems and applications for management", *Proc. Of the IEEE Global Telecommunications Conference*, pp. 107-111, Singapore, November 1995.

[6] William Stallings, *SNMP, SNMP-2, and CMIP, The practical Guide to Network Management Standards*, Addition-Wesley Publishing company, INC, 1993.

[7] M. Rose and K. McCloghrie, Structure and Identification of Management Information for TCP/IP-based Internets, May 1990, RFC 1155.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP), March 1990, RFC 1157.

[9] M. Rose and K. McCloghrie, Concise MIB Definitions, March 1991, RFC 1212.

[10] K. McCloghrie and M. Rose, Management Information Base for Network Management of TCP/IP-based internets: MIB-II, March 1991, RFC 1213.

[11] J. Galvin, K. McCloghrie and J. Davin, SNMP Security Protocols, July 1992, RFC 1352.

[12] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Introduction to Community-based SNMPv2, January 1996, RFC 1901.

[13] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1902.

[14] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1903.

[15] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1904.

[16] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1905

[17] .J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1906.

[18] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996, RFC 1907.

[19] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework, January 1996, RFC 1908.

[20] ISO/IEC, CCITT, Information Technology-OSI, OSI Basic Reference Model Part 4: Management Framework ISO/IEC 7498-4, CCITT Recommendation X.700, 1991.

[21] ISO/IEC, CCITT, Information Technology-OSI, Common Management Information Protocol (CMIP)-Part 1: Specification ISO/IEC 9596-1, CCITT Recommendation X.711, 1991.

[22] ISO/IEC, CCITT, Information Technology-OSI, Systems-Management Functions

Part1 – 14 ISO/IEC 10164-1 – 14, CCITT Recommendation X.730, X.732-737, X.739, X.740-742, 1991.

[23] ISO/IEC, CCITT, Information Technology-OSI, Management Information Model Part 1 – 2 ISO/IEC 10165-1– 2, CCITT Recommendation X.720 – 721, 1991.

[24] ISO/IEC, CCITT, Information Technology-OSI, Management Information Model Part 4: Guidelines for the Definition of Managed Objects ISO/IEC 10165-4, CCITT Recommendation X.722, 1991.

[25] ISO/IEC, CCITT, Information Technology-OSI, Elements of Management Information Related to OSI Network Layer Standards ISO/IEC 10733, 1991.

[26] OSIMIS Home Page, http://www.cs.ucl.ac.uk/research/osimis/

[27] G. Pavlou, Graham Knight, Kevin McCarthy, and Saleem Bhatti, "The OSIMIS Platform: Making OSI Management Simple", *Proc. of 4th IFIP/IEEE ISINM*, pp. 480-493, 1995.

[28] DSET Home Page, http://www.dset.com/ .

[29] IBM TMN Family of Products Home Page, http://issc2.boulder.ibm.com/telmedia/tmnbase.htm.

[30] Grillo, P., and S. Waldbusser, Host Resorces MIB, RFC 1514, Network Innovations, Intel Corporation, Canegie Mellon University, September 1993, http://www.simple-times.org/pub/simple-times/html/HOST-RESOURCES-MIB.

[31] Kille, S., and N. Freed, Network Services Monitoring MIB, RFC 1565, ISODE Consortium, Innosoft, January 1994, http://ds.internic.net/rfc/rfc1565.txt.

[32] C. Krupczak and J. Saperia, Definitions of System-Level Managed Objects for Applications, http://ds.internic.net/internet-drafts/draft-ietf-applmib-sysapplmib-08.txt, April 15, 1997. Internet Draft.

[33] Kalbfleisch, C., C. Krupczak and J. Saperia, Application Management MIB, 21 March, 1997, http://ds.internic.net/internet-drafts/draft-ietf-applmib-mib-02.txt .

[34] Kille, S., and N. Freed, Mail Monitoring MIB, RFC 1566, August 1996, http://ds.internic.net/rfc/rfc1566.txt .

[35] Mansfield, G., and S. Kille, X.500 Directory Monitoring MIB, AIC Systems Laboratory, ISODE Consortium, June 1996, http://ds.internic.net/rfc/rfc1567.txt.

[36] C. Kalbfleisch and H. Hazewinkel, Definition of Managed Objects for WWW Servers, OnRamp Technologies, Joint Research Centre of the E. C., January 1997, http://http-mib.onramp.net/draft/draft-ietf-applmib-wwwmib-00.txt.

[37] J. Schönwälder and M. Toet, "Management of the World Wide Web", *Proc. of Distributed Systems Operations & Management*, Sydney, Australia, pp. 42-52, October 1997.

[38] MAScOTTE, MAScOTTE Project (Management Services for Object oriented disTributed sysTEms) Esprit Project: 20804, White Paper, November, 1996. http://www.esrin.esa.it/ htdocs/MAScOTTE/.

[39] Monitoring of CORBA-based Applications, The CORBA-Assistant, White Paper, Fraunhofer Institute Informations and Date Processing IITB, May, 1997. http://tes.iitb.fhg.de/corba-assistant/.

[40] OMG, CORBAservices: EventService Specification, http://www.omg.org/corba/sectrans.htm, March 1995.

[41] IONA, OrbixTalk™ White Paper, IONA Technologies Ltd., April 1996.

[42] T. H. Yun, J. Y. Kong, and J. W. Hong, "Object-oriented modeling of distributed multimedia services", *Proc. of IEEE International Conference on Communications*, Montreal, Canada, pp. 777-781, June 1997.

[43] IONA, OrbixWeb, IONA Technologies Ltd., http://www.iona.com/Orbix/OrbixWeb/index.html, December 1996. Release 2.0.

[44] Subrata Mazumdar, WEB Based Management - CORBA/SNMP Gateway Approach , *Proc. of the Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, L'Aquila, Italy, October 1996.

[45] Subrata Mazumdar, "Inter-domain Management - CORBA, OSI, SNMP" , (Tutorial), *Proc of IM'97*, San Diego, May 1997.

# 부록 1: 관리 서버 & 관리 인터페이스 IDL

### ✍ common.idl

```
typedef string OID;
typedef short SID;
typedef short HID;
typedef long Token;
typedef short EventType;
typedef float PerfData;
typedef short PerfMetric;

typedef sequence<SID> SIDSeq;
typedef sequence<Token> TokenSeq;
typedef sequence<EventType>
    EventTypeSeq;
typedef sequence<string> stringSeq;
typedef sequence<long> longSeq;
typedef sequence<PerfData> PerfDataSeq;

struct AttrType {
   OID oid;
   string name;
   any value;
};

struct AttrType {
    OID oid;
    string name;
    any value;
};

typedef sequence<AttrType> AttrTypeSeq;

struct EventInfo {
    SID source;
    EventType type;
    short priority;
    TimeStamp time;
    string<100> description;
};

typedef sequence<EventInfo>
EventInfoSeq;
```

### ✍ cmso.idl

```
interface cMSO : ObjectMIO
{
boolean get_domain_info(in Token
    my_token, out string domain_name,
    out short num_hosts);
boolean get_host_info(in Token my_token,
    in HID host_id, out string
    host_name, out short host_status);
boolean detect_so(in Token my_token,in
    string domain_name,out SIDSeq sids);
boolean list_so(in Token my_token, in HID
    host_id, out SIDSeq sids);
SIDSeq search_so(in Token my_token, in
    string so_name)

string get_so_name(in Token my_token, in
    SID sid);
short get_so_status(in Token my_token, in
    SID sid);
SID get_sid(in Token my_token, in string
    host_name, in string so_name);

boolean get_attr(in Token my_token, in SID
    sid, in OID oid, out AttrType attr);
boolean set_attr(in Token my_token, in SID
    sid, in AttrType attr);
boolean get_attrs(in Token my_token, in
    SID sid, out AttrTypeSeq attrs);
short get_table_attrs(in Token my_token,
    in SID sid, in OID oid, out
    AttrTypeSeq attrs);

boolean init(in Token my_token, in SID
    sid);
boolean terminate(in Token my_token, in
    SID sid);

SID register_so(in string so_name, in
    string object);
boolean unregister_so(in Token my_token,
    in SID sid);
};
```

### ✍ fmso.idl

```
interface Efilter : ObjectMIO
{
boolean set_name(in string name);
string get_name();
boolean set_scope(in SIDSeq SIDs);
SIDSeq get_scope();
boolean set_eventlist(in EventTypeSeq
    eventtypes);
EventTypeSeq get_eventlist();
boolean set_eventinfo(in char infoset);
char get_eventinfo();
boolean start();
boolean stop();
string get_result(out EventInfoSeq
    eventinfos);
};

interface Elfilter : Efilter : ObjectMIO
```

```
{

boolean set_logfile(in string file);
string get_logfile();
};

interface fMSO : ObjectMIO
{
ELfilter create_elfilter(in Token
    my_token);
ERfilter create_erfilter(in Token
    my_token);
boolean destroy_efilter(in Token
    my_token, in Efilter efilter);
};
```

### ✍ pmso.idl

```
interface pMSO : ObjectMIO
{
oneway void start(in Token my_token);
boolean stop(in Token my_token);

boolean set_perf_test(in Token
    my_token,in SID sid);
boolean reset_perf_test(in Token
    my_token,in SID sid);
boolean list_tested_servers(in Token
    my_token, out SIDSeq sids);

PerfData get_recent(in Token
    my_token,in SID sid,in PerfMetric
    metric);
PerfData get_average(in Token
    my_token,in SID sid,in PerfMetric
    metric);
PerfData get_maximum(in Token
    my_token,in SID sid,in PerfMetric
    metric);
PerfData get_minimum(in Token
    my_token,in SID sid,in PerfMetric
    metric);
PerfDataSeq get_history(in Token
    my_token,in SID sid,in PerfMetric
    metric,in short num);

boolean set_threshold(in Token
    my_token,in SID sid,in PerfMetric
    metric,in PerfData threshold);
PerfData get_threshold(in Token
    my_token,in SID sid,in PerfMetric
    metric);
};
```

### ✍ smso.idl

```
interface sMSO : ObjectMIO
{
boolean add_user(in  Token my_token, in
    string id, in string passwd,in short
    authorization );
boolean delete_user(in Token my_token, in
    string id);
boolean change_passwd(in Token my_token,
    in string oldpasswd, in string
    newpasswd);

long login(in string id, in string
    passwd);
boolean unlog(in Token my_token);

boolean set_ACL(in Token my_token,in
    string id, in SID rid, in char
    permission);
char get_ACL(in Token my_token, in string
    id, in SID rid);

boolean is_authuser(in Token token);
string get_id(in Token token);

short list_users(in Token my_token, out
    string users);
boolean list_current_users(in Token
    my_token, out TokenSeq tokens);
};
```

### ✍ mio.idl

```
interface BasicMIO
{
readonly attribute short attr_count;
boolean get(in OID oid, out AttrType
    attr);
boolean set(in AttrType attr);
boolean gets(out AttrTypeSeq attrs);
short get_table(in OID oid, out
    AttrTypeSeq attrs);
};

interface ObjectMIO : BasicMIO {
};

interface MIO : BasicMIO
{
boolean init();
}
```