통합 망 관리 에이전트 설계 및 구현

박 상 철[†]·김 태 수[††]·이 광 휘[†††]

요 약

본 논문에서는 인터넷 망을 위한 SNMP(Simple Network Management Protocol)를 이용하는 망 관리 시스템과 OSI 망을 위한 CMIP (Common Management Information Protocol)을 사용하는 망 관리 시스템을 상호 연동할 수 있는 통합 망 관리 에이전트 (agent)를 설계, 구현하였다. 서로 다른 두 망 관리 시스템을 통합하기 위하여 본 연구에 앞서 수행한 MOVI(Managed Object View Interface) 개념을 이용하였다. 통합 망 관리 에이전트 설계를 위한 3가지 모델을 검토하고 이 모델들의 장단점을 비교하여 적합한 모델을 선택 하였다. 선택한 모델을 구체화 할 수 있는 통합 망 관리 에이전트의 논리적 구조와 구체적인 구현방법 그리고 각 모듈별 작동에 대하여 언급하였다. 본 연구에서는 기존의 망 관리 시스템인 osimis와 ucd-snmp 시스템을 참조 모델로 이용하였다. 본 연구를 통하여 인터넷 망과 OSI 망을 망 관리적 측면에서 연동할 수 있을 뿐 아니라 앞으로 MOVI 개념에서 언급하고 있는 것과 같이 새롭게 정의될 수 있는 망 관리 시스템과의 연동에 대한 확장도 가능하다는 장점을 가진다.

Design and implementation of an integrated network management agent

Sang-Cheol Park[†] · Tae-Soo Kim[††] · Kwang-Hui Lee[†††]

ABSTRACT

In this paper, an integrated network management agent has been proposed and implemented to support different network management protocols, SNMP (Simple Network Management Protocol) for Internet and CMIP (Common Management Information Protocol) for OSI networks. We used MOVI (Managed Object View Interface) concept to integrate the different network management systems. We reviewed three models to design the integrated network management agent and then selected a suitable model among them. The logical structure of an agent, the implementation method and the operation of each module have been shown in this paper. The osimis and ucd-snmp network management system have been used as the reference systems for implementing our system. The integrated network management agent can support the internetworking between Internet and OSI networks in the aspect of network management. Using MOVI concept, if a new management system is introduced, internetworking with this can be achieved by adding the new interface on the view interface of managed object.

키워드 : 망 관리(Network Management), CMIP, 단순망관리프로토콜(SNMP), 관리정보베이스(MIB), MOVI, 에이전트(Agent)

1. 서 론

다양한 통신망이 개발되어 사용되고 있는 현재의 환경에서 서로 다른 특성을 갖는 통신망의 연동은 중요한 연구분야 중의 하나이다. 연동을 이루기 위하여는 프로토콜의 스택 상에서 여러 계층에서 연동이 이루어져야 한다. 하위 계층들에서는 이와 관련된 연구가 활발하게 진행되어 왔다. 물론 상위의 응용 계층에서도 응용 게이트웨이 (gateway)들이 개발되었다. 본 연구에서는 이러한 것 중에서 응용 계층에 있는 망 관리 기능의 연동과 관련된 사항에 대하여 언급하도록 한다.

※ 이 연구는 정보 통신부에서 지원하는 대학 기초 연구 지원 사업 (1999 2001)결과의 일부임.
† 정 회 원 : 창원대학교 대학원 컴퓨터공학과
†† 정 회 원 : 창원대학교 대학원 컴퓨터공학과
††† 정 회 원 : 창원대학교 컴퓨터공학과 교수
논문접수 : 2001년 3월 16일, 심사완료 : 2001년 7월 26일

현재 널리 알려진 망 관리(network management)는 SNMP (Simple Network Management Protocol) 프로토콜을 이용하는 인터넷(internet) 망 관리와 CMIP(Common Management Information Protocol) 프로토콜을 이용하는 OSI 망 관리로 나누어 생각할 수 있다. 일반적으로 SNMP 프로토콜로 알려진 인터넷 망 관리 시스템은 인터넷을 기반으로 하여 많이 사용되고 있고, 구조가 비교적 간단하므로 사용과 구현이 용이한 특성을 가지고 있다. 반면에 OSI 망 관리 시스템은 개념이 잘 정의되어 있으나 구현이 복잡하고 사용할 때 부하가 크다는 특성을 가지고 있다[1]. 전체적으로 인터넷이 널리 사용되고 있기는 하지만 OSI 망 또한 전화망 관리에서는 부분적으로 사용되고 있다. 그러므로 인터넷 망 관리와 OSI 망 관리를 서로 연동하는 경우 적용 범위를 넓힐 수 있다는 장점을 가지게 된다. 또한 관리의 대상이 되는 객체가 두 망 관리 시스템에 의하여 접근되는

경우에는 연동 시스템이 필요하게 된다[2].

따라서, 본 논문에서는 이들 두 망 관리 시스템을 상호 연동할 수 있는 통합 에이전트에 대하여 언급한다. 특히 이미 많이 연구된 기존의 통합 망 관리 시스템과 구별되는 다른 개념을 사용하여 연동을 이루고자 한다. 즉, 통합과 연동에 대한 대상을 프로토콜 변환 방법이 아닌 관리객체가 서로 다른 망 관리 프로토콜을 지원할 수 있도록 하기 위하여 MOVI(Managed Object View Interface)를 이용한다[5]. 또한 본 논문에서는 이미 선행된 연구를 통하여 새롭게 정의된 관리객체를 지원하기 위한 통합 에이전트의 설계와 구현에 중점을 두도록 하였다[3].

2장에서는 먼저 MOVI 개념에 대하여 간략히 언급하고 통합 망 관리 에이전트의 설계를 위한 3가지 방법에 대하여 설명하도록 한다. 통합 망 관리 에이전트의 구조 및 구현 그리고 실행 예에 대하여는 3장에서 보인다. 그리고 4장에서 간략하게 결론을 맺도록 한다.

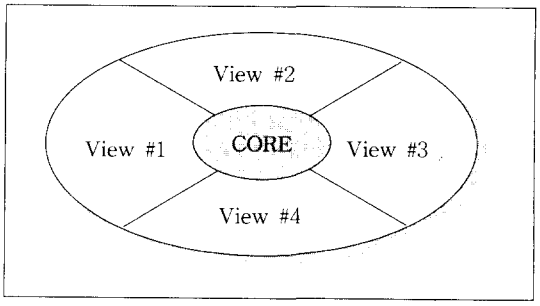
2. 시스템 모델

본 장에서는 본 연구와 관련된 MOVI 개념에 대하여 먼저 설명하도록 한다. 그리고 망 관리 시스템의 연동을 위한 에이전트를 설계 및 구현하기 위한 방법에 대하여 언급하도록 한다. MOVI 개념은 하나의 관리 대상객체를 서로 다른 두 망 관리 시스템이 접근할 수 있도록 하기 위한 방법이다. MOVI에 의하여 정의된 관리대상 객체를 접근하기 위하여 에이전트가 필요하다. 이 에이전트를 구현하기 위한 방법을 세가지로 검토하였다.

2.1 MOVI(Managed Object View Interface)

서로 다른 망 관리 시스템, 즉, 관리자(manager) 시스템과 에이전트(agent) 시스템이 다른 망 관리 시스템을 사용하고 있는 경우 이 두 시스템을 통합하는 방법은 일반적으로 많이 알려진 방법인 프로토콜 게이트웨이(protocol gateway)를 이용하는 방법과 관리객체를 변환하는 방법인 IIMC (ISO/CCITT and Internet Management Coexistence)를 이용하는 방법이 있다[4].

본 논문에서는 관리객체에 초점을 맞춘 MOVI라는 개념을 이용한다[3]. MOVI는 (그림 1)과 같이 관리객체를 Core 부분과 View 부분으로 나누어 구성한다. 즉, Core 부분은 관리대상 객체의 공통적인 부분으로 일관성이 유지되게 구성하고, View 부분은 각 관리자에게 다르게 보여지는 부분으로 망 관리 시스템에 따라 각기 다르게 구성된다. 서로 다른 망 관리 프로토콜이 관리대상 객체를 접근하는 경우 관리대상 객체는 두 망 관리 시스템에게 서로 자신의 프로토콜을 사용하는 관리대상 객체인 것으로 인식될 수 있도록 해줄 필요가 있다. 이러한 것은 관리대상 객체의 View 부분이 제공하게 된다. 그리고 관리대상 객체는 자신의 고유한 관리정보는 Core 부분에 위치하도록 한다. 그리하여 망 관리 시스템은 관리대상 객체를 자신의 프로토콜을 사용하여 접근할 수 있다. 이와 같이 관리객체를 이중적으로 구성함으로써 관리객체에 접근하는 관리자가 바뀌게 되어도 관리객체에 대한 인터페이스인 View 부분을 새롭게 구성함으로써 확장이 가능하다.



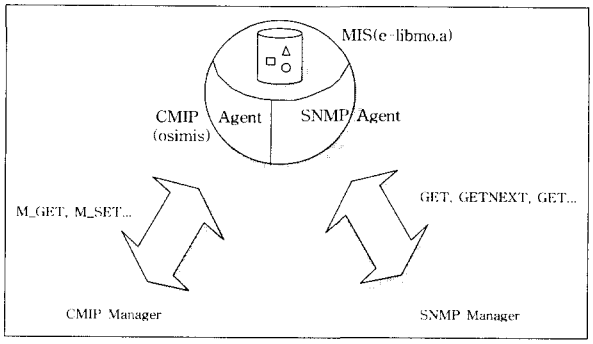
(그림 1) MOVI(Managed Object View Interface) 개념도

2.2 통합 망 관리 에이전트 설계를 위한 3가지 모델

기존의 망 관리 시스템인 osimis와 ucd-snmp 시스템의 동작과 구조를 고려할 때 통합 에이전트의 구조는 3가지 모델로 개발이 가능하다[6-10]. 첫째로 기존의 osimis 시스템에 SNMP 에이전트의 기능을 추가하는 1-에이전트 프로세스 모델(그림 2), 둘째로 기존의 CMIP 에이전트(osimis 시스템)와 SNMP 에이전트(ucd-snmp 시스템)를 최소한으로 수정하면서 통합하는 2-에이전트, 1-중재자 프로세스 모델(그림 3) 그리고 마지막으로 내부에 있는 MIB를 외부의 별도 프로세스로 분리한 2-에이전트, 1-MIB 프로세스 모델(그림 4)을 생각할 수 있다. 이들 각각의 특징을 아래에서 언급하도록 한다.

2.2.1 1-에이전트 프로세스 모델

이 모델은 기존의 osimis 시스템에 SNMP 에이전트의 기능을 추가하는 모델이다. 즉, CMIP 에이전트와 SNMP 에이전트, 그리고 MIB까지도 하나의 프로세스에 존재하게 되는 구조이다. 내부구조는 (그림 2)과 같이 하나의 에이전트에서 두 종류의 망 관리 프로토콜을 지원하는 형태가 된다.



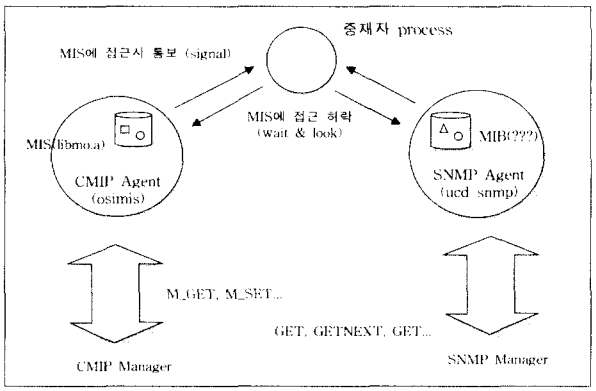
(그림 2) 1-에이전트 프로세스 모델

이 모델의 경우 두개의 서로 다른 프로토콜 스택(OSI와 TCP/IP)을 동시에 지원해야 되는 것과 기존의 OSIMIS 시스템에 SNMP 에이전트 기능을 추가한다는 것이 구현상 어려우며, 프로세스의 인터페이스 부분이 복잡하게 될 것이다.

2.2.2 2-에이전트, 1-중재자 프로세스 모델

이 모델은 기존의 CMIP과 SNMP 에이전트를 최소한으로 수정하면서 MOVI 개념을 구현할 수 있는 현실적인 방법이라고 할 수 있다. 이 모델의 내부구조는 (그림 3)과 같이 기존의 두 에이전트들 각각이 독립적으로 MIB를 가지고 있고, 이 두 에이전트들의 MIB에 대한 접근을 중재자(Coordinator) 프로세스가 관리 및 통제하고 있는 모델이다. 즉, 각 에이전트는 MIB에 대한 접근을 위한 승인을 중재자 프로세스를 통해서 획득하고 승인을 얻은 경우에만 프로세스 내부의 MIB에 대한 접근을 한다. 또한 중재자 프로세스는 각 에이전트의 관리객체에 대한 접근 정보를 일괄적으로 관리 및 유지하기위해 관리객체 참조 테이블을 운영한다.

이 모델은 기존의 CMIP 에이전트와 SNMP 에이전트에 대한 수정 많지 않다는 장점을 가지고 있는 것으로 생각될 수 있다. 그러나 중재자를 설계 및 구현하기가 복잡하고 실제 MIB에 대한 접근의 효율성이 저하될 수 있는 단점을 가지고 있다. 또한 두 곳에 존재하는 MIB의 일치성을 유지하는 것이 어려운 문제이다.

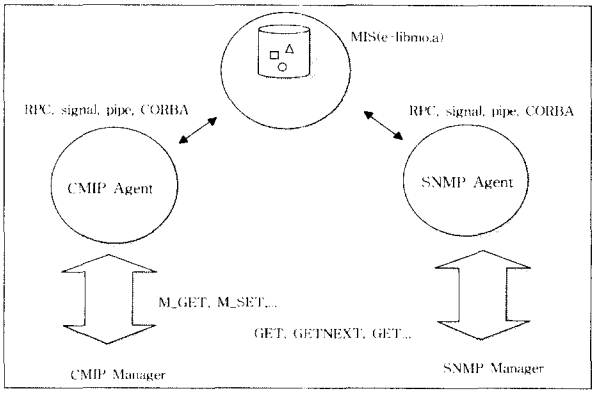


(그림 3) 2-에이전트, 1-중재자 프로세스 모델

2.2.3 2-에이전트, 1-MIB 프로세스 모델

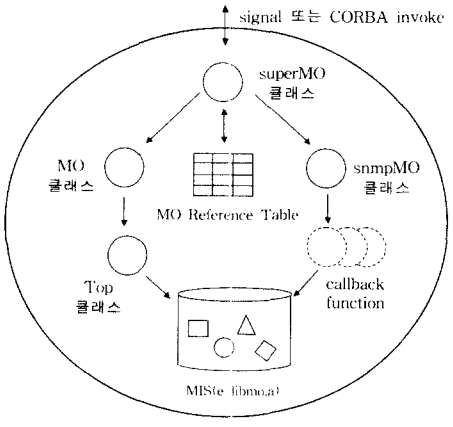
MOVI 개념을 가장 충실하게 표현할 수 있는 모델인 이 구조는 각 에이전트에서 MIB에 해당되는 부분을 별도의 MIB 프로세스로 분리하는 형태이다. 내부구조는 (그림 4)와 같이 CMIP 에이전트와 SNMP 에이전트 그리고 MIB 프로세스가 독립된 프로세스로 존재한다. 각 에이전트에서 분리된 MIB는 통합된 형태로 하나의 MIB 프로세스에 존재하게 된다.

E-GDMO(Extended GDMO)로 기술된 관리객체에 대한 정의는 E-GDMO 컴파일러를 통해서 번역되어 각각 CMIP과 SNMP부분으로 분리되어 각각 GDMO와 ASN.1으로 기술된 관리객체에 대한 정의를 생성한다[3, 5, 8, 15]. 그리고 생성된 정의는 OSIMIS와 UCD-SNMP의 기존 방법과 동일하게 각각 GDMO와 ASN.1 컴파일러를 통해서 관리객체에 대한 메타 클래스가 생성되고 이 각각의 메타 클래스가 링크 과정을 통해서 MIB 프로세스에 포함되게 된다. 그러므로 MOVI의 개념과 같이 서로 다른 두개의 에이전트가 하나의 MIB를 공유하면서 접근하게 되는 모델이 된다.



(그림 4) 2-에이전트, 1-MIB 프로세스 모델

MIB 프로세스의 내부구조를 보면 (그림 5)와 같이 CMIP과 SNMP의 기존 MIB구조를 새로운 superMO클래스를 이용해서 일관되게 접근하는 방법을 제시하고 있다. 뿐만아니라 이 새로운 superMO클래스는 외부의 signal이나 CORBA의 invoke를 처리하는 모듈을 통해서 MIB에 대한 일관된 접근을 허락한다. 또한 MIB 프로세스는 각 에이전트의 관리객체에 대한 접근을 통제하고, 관련 접근 정보(관리객체 참조 테이블)를 관리, 유지한다. 이 구조는 세 개의 프로세스를 어떻게 효율적으로 연결하느냐가 중요한 문제이다. 각각의 프로세스는 RPC, CORBA 등의 방법을 이용해서 연결모듈을 구현할 수 있다.



(그림 5) MIB 프로세스의 내부구조

2.2.4 각 모델별 비교

본 논문에서 고찰한 3가지 모델을 <표 1>로 정리하였다.

〈표 1〉 각 모델별 비교

비교항목 / 모델	MOVI 개념 표현	기존 에이전트 수정 여부	확장성	안정성	구현상의 어려움
1-에이전트	부 족	적 음	부 족	부 족	어렵다
2-에이전트, 1-중재자	보 통	적 음	우 수	보 통	어렵다
2-에이전트, 1-MIB	우 수	보 통	우 수	우 수	미들웨어 사용가능

1-에이전트 모델은 한 시스템 내에 서로 다른 프로토콜 스택이 동시에 지원되어야 하므로 구현상의 문제 뿐만아니라 확장성 및 안정성면에서도 적절하지 않다.

2-에이전트, 1-중재자 프로세스 모델은 MOVI개념을 우회적으로 표현하고 있지만 기존 에이전트 시스템에 대한 수정이 적은 장점을 가지며 확장성이 우수하다. 하지만 중재자를 구현하고 여러 시스템에 존재하는 MIB의 일치성을 유지하는 것이 어려운 문제이다.

2-에이전트, 1-MIB 프로세스 모델은 MOVI개념을 가장 이상적으로 표현할 수 있으며, CORBA와 같은 미들웨어의 사용이 가능하여 구현상의 부하가 적은 장점을 가지고 있다. 또한 이동통신망과 같은 새로운 망 관리 프로토콜에 대한 확장성에서 다른 모델들 보다 우수하다. 따라서 본 논문에서는 이 모델을 채택하였다.

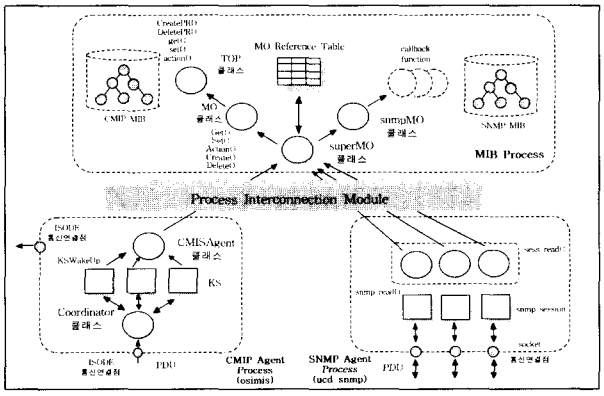
3. 통합 망 관리 에이전트 구현 및 실행 예

3.1 통합 망 관리 에이전트의 논리적 구조

2장에서 살펴본 바와 같이 세 번째 모델인 2-에이전트, 1-MIB 프로세스 모델을 기반으로 하여 기존의 osimis 시스템과 ucd-snmp 시스템을 통합한 통합 에이전트의 내부 구조는 (그림 6)과 같다. 각 모듈의 특성 및 동작에 대하여 언급하도록 한다.

3.1.1 MIB 프로세스 모듈

MIB 프로세스 모듈은 각 에이전트에 있던 MIB를 관련된 클래스 또는 함수와 함께 별도의 프로세스로 분리하여 하나의 모듈로 만드는 것이다. osimis 시스템의 CMIP 에이전트인 경우에는 MIB와 관련이 있는 클래스인 MO, Top 클래스 등과 MIB 자체가 포함되고, ucd-snmp 시스템의 SNMP 에이전트인 경우에는 각 MIB에 접근하기 위해서 생성되는 callback 함수와 MIB 자체가 포함되었다. 뿐만 아니라 서로 다른 MIB에 대한 접근을 조정하기 위해서 몇 개의 클래스와 부 모듈(sub module)들이 추가되었다. MIB 프로세스 모듈의 부 모듈(sub module)들에 대하여 설명하도록 한다.



(그림 6) 통합 망 관리 에이전트의 논리적 구조

1) 관리객체 참조 테이블(MO Reference Table)

(그림 6)에서 볼 수 있는 것과 같이 MIB 프로세스 내부에는 서로 다른 MIB, 즉 CMIP과 SNMP의 MIB가 같이 존재한다. 또한 몇 관리객체는 서로 공유되기도 한다. 그러므로 이들 관리객체, 특히 서로 공유되고 있는 관리객체(shared managed object)에 대한 접근을 통제하기 위해서 관리객체 참조 테이블(MO Reference Table)을 두고 있다. 이 관리객체 참조 테이블을 통해서 필요한 관리객체에 대한 접근을 통제하고 기록하게 된다. 또한 이 관리객체 참조 테이블은 superMO클래스를 통해서 참조되므로 일관된 접근과 통제가 가능하게 된다. 관리객체 참조 테이블에서의 동작과정을 의사코드(pseudo code)로 표현하면 다음과 같다.

```
a. MOItem old_MO = findMO(current_MO);
b. if (old_MO != null) {
        if (current_MO.hasMutualOperation() ||
            old_MO.hasMutualOperation()) {
            current_MO.waitForAccess(MOItem :: INFINITE);
        }
   }
c. moRefTable.accessMIB(current_MO);
d. current_MO.runOperation();
e. moRefTable.deaccessMIB(current_MO);
```

그리고 각 과정별 절차를 구체적으로 설명하면 다음과 같습니다.

a. 현재 접근하려는 MO(current_MO)와 동일한 MO가 이미 MIB에 접근하고 있는지를 확인한다.

b. old_MO 값이 null이 아니면 즉, current_MO와 동일한 MO가 이미 MIB에 접근하고 있다면 현재 접근하려는 MO(current_MO)의 operation이 배타적인 operation (SET operation)이거나 과거에 이미 MIB에 접근하여 사용하고 있는 old_MO의 operation이 배타적인 operation(SET operation)이면 현재 접근하려는 MO(current_MO)는 배타적인 접근(mutual exclusion)이 해제될 때까지 기다린다. current_MO와 old_MO가 같다는 것은 MIB내의 접근하는 MO가 같지만 이 MO에 접근하는 operation은 다를 수 있다. 즉, 처음에는 GET operation으로 접근하고 두 번째는 SET operation으로 접근할 수 있다. 여기서 waitForAccess() 메소드는 Mutex 객체를 이용하여 MIB에 대해서 상호 배타적인

접근을 통제하는 메소드이다.

c. moRefTable에 접근하려는 MO에 대한 정보를 기록한다. 그리고 동일한 MO에 대한 접근을 통제하기 위하여 MOItem :: ccessMutex.lock()을 호출하여 Mutual exclusion을 설정한다.

d. 접근하려는 MO에 대해서 실제적인 operation(SET 또는 GET)을 수행한다.

e. 이미 기록된 MO에 대한 정보를 moRefTable에서 삭제한다. 그리고 동일한 MO에 대한 접근을 허락하기 위하여 MOItem :: accessMutex.release()을 호출하여 Mutual exclusion을 해제한다.

2) superMO 클래스

(그림 6)의 통합 에이전트의 논리적인 구조에서 볼 수 있는 것과 같이 superMO 클래스는 MIB 프로세스의 핵심적인 클래스라고 할 수 있다. 앞에서 언급한 관리객체 참조 테이블 또한 이 클래스를 통해서 접근이 가능하고 두 에이전트도 MIB에 접근하기 위해서는 이 클래스를 이용한다. superMO 클래스의 기본 기능을 언급하면 다음과 같다.

a. 각 에이전트에서 하위의 MO 클래스와 snmpMO 클래스(callback 함수)에 접근할 수 있도록 한다. 클래스 또는 함수에 대한 포인터를 전달한다.

b. 관리객체 참조 테이블을 이용해서 접근하는 관리객체에 대한 접근을 통제, 조정한다.

c. 프로세스 연결 모듈(Process Interconnection Module)과의 인터페이스 역할을 한다. CORBA의 idl 인터페이스 또는 RPC의 stub 함수를 제공한다.

또한 각 에이전트들과 프로세스 연결 모듈을 통해서 연결되기 위해서 주고 받아야 할 값(value)들은 다음과 같다.

a. MIB 프로세스에서 각 에이전트에 전달되어야 할 값(value)들
- MO 클래스의 MIB에 접근할 수 있는 멤버함수에 대한 포인터 또는 stub 함수(MO :: Get(), MO :: Set(), MO :: Action(), MO :: Create(), MO :: Delete() 등)
- callback 함수에 대한 포인터 또는 전달하기 위한 stub 함수

b. 각 에이전트에서 MIB 프로세스에 전달되어야 할 값(value)들
- MO 객체(관리객체 이름, oid, attribute 등) 또는 MO 객체에 대한 포인터, 수신된 PDU 등
- 수신된 PDU, snmp_session에 대한 포인터 등

3) snmpMO 클래스

snmpMO 클래스 또한 superMO 클래스와 마찬가지로 새롭게 첨가된 클래스이다. 그렇지만 superMO 클래스에 비하면 간단하다. snmpMO 클래스의 가장 주요한 기능은 SNMP MIB에 접근할 수 있는 callback 함수들에 대한 포인터를 저장, superMO 클래스를 통해서 SNMP 에이전트에 전달하는 것이다. callback 함수 포인터를 전달하는 방법을 간략하게 보면 다음과 같다.

a. CORBA를 이용하여 프로세스 연결 모듈을 구성할 경우
- 각각의 callback 함수에 대한 포인터를 callback_var라는 변수에 저장한다.
- superMO 클래스를 통해서 callback_var라는 변수값을 전달한다.

b. RPC를 이용하여 프로세스 연결 모듈을 구성할 경우
- callback 함수에 대한 포인터를 전달하기 위한 pass_callback() 함수를 정의한다.
- pass_callback() 함수를 이용해서 callback 함수에 대한 포인터를 전달한다.

이와 같이 callback 함수에 대한 포인터를 전달하는 방법이 프로세스 연결 모듈을 구성하는 방법에 따라 다른 이유는 RPC를 이용하는 경우에 특정 변수값을 함수를 이용하지 않고는 전달할 수 없기 때문이다. 자세한 내용은 3.2절의 프로세스 연결 모듈 구현에서 다루도록 한다.

4) MIB 접근 부 모듈

각 MIB에 대한 실제적인 접근은 본 시스템의 참조 모델 중의 하나인 osimis 시스템과 ucd-snmp 시스템과 거의 동일하다. 내용을 간략하게 설명하면 다음과 같다.

a. CMIP MIB에 대한 접근
- 모든 관리객체는 Top 클래스로부터 상속된다.
- Top 클래스의 멤버 함수인 get(), set(), action(), CreateRR(), DeleteRR() 등의 함수를 이용해서 실제 MIB에 일관되게 접근한다. 가상함수의 상속이므로 자동으로 생성된다.

b. SNMP MIB에 대한 접근
- 관리객체에 대한 MIB를 생성할 때 이 관리객체에 접근하기 위한 callback() 함수를 함께 정의한다.
- callback 함수에 대한 포인터는 항상 동일한 형(type)을 가진다. 형은 다음과 같다.

```
int (*)(int, struct snmp_session *,
        int, struct snmp_pdu *, void *)
```

지금까지 MIB 프로세스 모듈에 대하여 설명하였으며 이제는 통합 에이전트의 중요한 부분 중의 하나인 프로세스 연결 모듈에 대해서 기능 중심으로 간략하게 알아보겠다. 실제적인 구현은 여러 가지가 있을 수 있으므로 구체적인 구현은 3.2절에서 설명한다.

3.1.2 프로세스 연결 모듈

2장에서도 언급한 것과 같이 본 연구에서 채택한 2-에이전트, 1-MIB 프로세스 모델은 기존 시스템(osimis, ucd-snmp)과 달리 MIB를 에이전트 프로세스 외부로 분리하게 된다. 그러므로 에이전트 프로세스와 분리된 MIB(MIB 프로세스)를 연결하기 위해서는 프로세스 연결 모듈이 필요하다. 프로세스 연결 모듈의 역할을 언급하면 다음과 같다.

a. 두 에이전트 프로세스(CMIP 에이전트, SNMP 에이전트 프로세스)와 MIB 프로세스를 투명하게 연결한다.
b. CMISAgent 클래스와 snmp_read() 함수에게 MO 클래스와 callback 함수에 대한 접근을 위해서 superMO 클래스에 대한 접근을 보장한다.
c. 관리객체에 대한 명령의 수행 결과를 MIB 프로세스에서 각 에이전트 프로세스에게 전달한다.

3.1.3 에이전트 통신 모듈

앞에서도 언급한 것과 같이 본 연구에서 구현한 통합 에이전트는 기존의 시스템인 osimis와 ucd-snmp 시스템을 근간으로 하고 있다. 그러므로 (그림 5)의 논리적인 구조에서 보인 바와 같이 각 에이전트의 통신 모듈은 기존 시스템의 것을 이용하고 있다.

1) CMIP 에이전트 통신 모듈

CMIP 에이전트는 ISODECoordinator 클래스의 readCommEndpoints() 함수(내부적으로 ISODE의 iserver_wait() 함수를 이용하고 있다)를 통해서 ISODE 통신연결점(connection point)으로부터 메시지가 수신되기를 기다리고 있다. 이것은 message listening 단계에서 이루어진다. 메시지가 수신되면 associationWork() 함수를 통해서 제어는 CMISAgent 클래스로 전달되고 수신된 메시지는 KS 클래스와 KSWakeUp 자료구조를 이용해서 연결된 association별로 관리되어 CMISAgent 클래스로 전달된다. CMISAgent 클래스에서는 앞에서도 고찰한 바와 같이 프로세스 연결 모듈과 superMO 클래스를 통해서 분리된 MIB에 접근하게 된다. 그리고 MIB에 대한 명령을 수행한 뒤 결과는 CMISAgent 클래스에서 ISODE 통신연결점을 통해서 외부로 전달된다.

2) SNMP 에이전트 통신 모듈

(그림 5)에서 볼 수 있는 것과 같이 socket 통신연결점을 통해서 수신된 메시지(PDU)는 수신된 세션 단위별로 관리되어 별도로 수신된 명령을 수행한다. 이 세션은 snmp_session이라는 구조체를 이용해서 정보가 저장되고 _sess_read() 함수를 통해서 개별적으로 PDU를 수신, 조립하여 순차적으로 프로세스 연결 모듈과 superMO 클래스를 통해서 분리된 MIB에 접근한다. 이상의 내용을 의사코드(pseudo code)로 표현하면 다음과 같다.

```
receive()          /* snmpd.c의 main() 함수의 메시지 수신부 */
{ .....
    while (running)    /* 무한 loop */
    {  .....
        for (session list 처음부터 마지막 요소까지)
                                        /* snmp_read() 함수 */
        {        /* snmp_session_read(), _sess_read() 함수 */
            accept();                   /* socket accepting */
            recv(); 또는 recvfrom();    /* 메시지(PDU) 수신 */
            snmp_parse();               /* PDU parsing */
            ......
            callback();   /* callback 함수를 통한 MIB 접근 */
        }
        .....
    }
}
```

지금까지 통합 에이전트의 논리적인 구조에 대하여 설명하였다. 다음에는 앞에서도 언급한 바가 있는 프로세스 연결 모듈의 구체적인 구현에 대해서 언급한다.

3.2 프로세스 연결 모듈 구현

(그림 5)의 통합 에이전트의 논리적인 구조에서도 언급한 것과 같이 통합 에이전트는 서로 다른 프로세스 3개가 상호 유기적으로 작동하고 있다. 그러므로 이 3개의 프로세스를 연결하는 프로세스 연결 모듈(Process Interconnection Module : PIM)의 구현이 중요하다. 3.1절에서 통합 에이전트의 구조와 함께 프로세스 연결 모듈의 기능에 대해서 설명하였으며 3.2절에서는 이 프로세스 연결 모듈의 구현에 대해서 언급한다.

구현은 3.1절에서도 간단하게 언급한 바와 같이 CORBA와 같은 미들웨어(middle-ware)를 이용한 방법과 RPC(Remote Procedure Call)를 이용한 방법, 그리고 Unix의 파이프(pipe), 시그널(signal), 공유메모리 등의 System Call을 이용한 방법[11], 그리고 Microsoft사의 Windows 시스템인 경우에는 동적 링크 라이브러리(DLL, Dynamic Link Library)를 이용한 방법 등이 있다[12].

그렇지만 이 방법들 중에서 세 번째의 Unix System Call을 이용해서 프로세스 연결 모듈을 만드는 것은 middle-ware 수준의 프로그램을 다시 만들어야 하므로 구현하기 어렵고 구현기간이 길다. 그리고 기존 middle-ware 시스템인 CORBA나 RPC등의 기존 제품을 이용하는 것보다 상대적으로 안정성이 낮아질 것이다. 또한 마지막의 동적 링크 라이브러리를 이용한 방법도 실제로 Windows 시스템의 SNMP 에이전트가 이 방법을 사용하고 있으나, 기존 시스템인 osimis와 ucd-snmp 시스템이 UNIX 환경에서 개발된 관계로 본 연구에서 채택하기에는 적합하지 않다. 그래서 본 절에서는 첫 번째와 두 번째의 CORBA와 Remote Procedure Call을 이용한 방법만 언급하도록 한다.

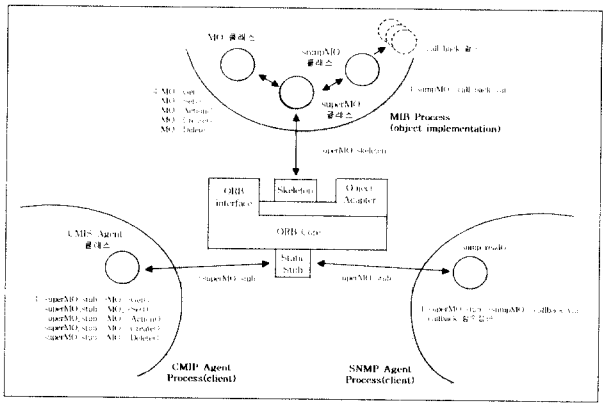
3.2.1 CORBA를 이용하는 경우

CORBA(Common Object Request Broker Architecture)

는 OMG(Object Management Group)에서 멀티벤더(multi-vendor)를 지원하는 객체 지향적인 분산 컴퓨팅을 지원하기 위한 표준 아키텍처이다[13].

CORBA 환경은 개별 객체들의 구체적인 구현이나 분산 환경의 변화와 상관없이 ORB와 IDL 인터페이스를 통해서 동일한 클라이언트의 애플리케이션 코드를 사용할 수 있도록 한다. 다시 말해서 CORBA 환경을 이용해서 시스템을 개발하는데 관건은 정확한 IDL 인터페이스의 작성이라 할 수 있다. 여기서 ORB는 IDL 인터페이스 뿐만 아니라 동적인 요청을 위한 인터페이스인 DII(Dynamic Invocation Interface), 외부의 다른 ORB와 연결하기 위한 DSI(Dynamic Skeleton Interface), 그리고 ORB 인터페이스, 오브젝트 어댑터(Object Adapter)등을 가지고 있다.

CORBA를 이용하는 경우 (그림 7)에서 보는 바와 같이 두 에이전트는 IDL 인터페이스를 통하여 MIB 프로세스의 MO 클래스와 snmpMO 클래스에 접근한다. CMIP 에이전트는 MO 클래스의 MIB 인터페이스 함수들(MO::Get(), MO::Set() 등)을 이용하고, SNMP 에이전트는 snmpMO 클래스의 멤버 변수인 snmpMO::callback_var와 callback 함수를 이용하여 MIB에 접근한다.



(그림 7) CORBA를 이용하는 경우 프로세스 연결 모듈 구조

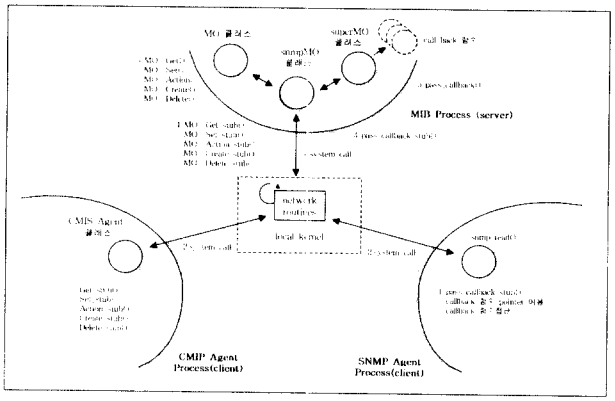
이상의 내용을 바탕으로 IDL 인터페이스를 기술하면 다음과 같다.

```
// superMO.idl
interface superMO {
    .....
    readonly attribute snmp_callback callback_var;
                        // snmpMO::callback_var
    .....
    MO* getMORoot();    // MO::getRoot();
    .....
};
```

3.2.2 RPC를 이용하는 경우

RPC는 단일 프로세스에서 호출되는 로컬 프로시저 호출(local procedure call)과 달리 특정 국부 시스템에서 원격 시스템에 있는 프로시저를 호출한다. 일반적으로 국부 프로시저 호출은 특정 부 프로시저로 제어를 넘기면 되지만 RPC는 완전히 다르게 작동한다[14].

RPC를 이용하는 경우 (그림 8)과 같이 세 프로세스는 생성된 stub 함수들과 자신의 커널을 통하여 상호 연결된 구조를 갖는다. 즉, CMIP 에이전트인 경우는 superMO 클래스를 통해서 접근하고자 하는 MO 클래스의 MIB 인터페이스 함수에 대응되는 stub 함수를 만든다. 그리고 SNMP 에이전트인 경우는 superMO 클래스와 snmpMO 클래스를 통해서 접근하고자 하는 pass_callback() 함수에 대한 stub 함수를 만든다.



(그림 8) RPC를 사용하는 경우 프로세스 연결 모듈 구조

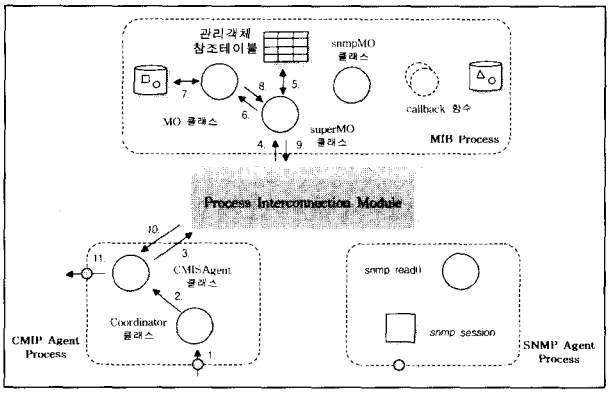
이렇게 생성된 함수는 서버측이 되는 MIB 프로세스와 클라이언트측이 되는 두 에이전트 프로세스간에 각각 생성되어서 원격 호출이 가능하도록 한다. 프로세스 연결 모듈의 구현에 대해서 설명하였으며, 다음 3.3절에서는 CMIP과 SNMP의 관리자(manager)에서 요청한 명령(operation)들이 통합 에이전트에서 어떻게 처리되는지를 설명한다.

3.3 시스템 동작 예

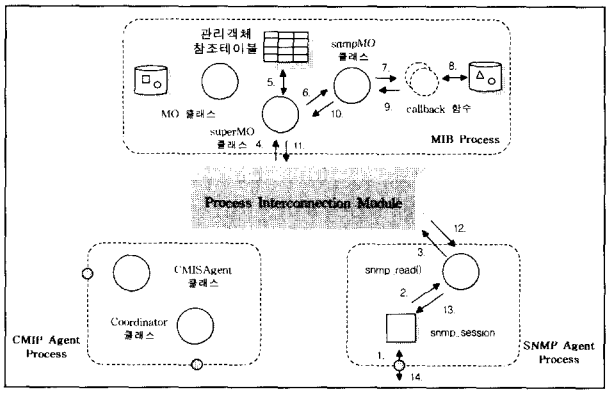
본 절에서는 각 관리자에서 요청한 명령들이 통합 에이전트에서 어떻게 처리되는지를 살펴보도록 한다. 대표적으로 네 가지의 경우로 나누어서 설명하도록 한다. 먼저, CMIP MIB에 대한 Get 명령, 다음은 SNMP MIB에 대한 Set 명령, 그리고 SNMP와 CMIP에 의하여 공유되는 관리 객체에 대한 Get, Set 명령을 수행하는 경우를 예로서 설명하도록 한다.

3.3.1 CMIP MIB에 대한 Get 명령

CMIP MIB에 대한 Get 명령을 수행할 때는 osimis 시스템의 동작 모습과 흡사하다. 기존의 osimis 시스템과 다른 점은 MIB 모듈이 다른 프로세스로 분리되어 있으므로 프로세스 연결 모듈을 통해서 MO 클래스와 MIB에 접근을 할 수 있다. (그림 9)에서 각 단계별 동작모습을 보였고 아래에 이에 대한 동작을 간단히 설명하였다.



(그림 9) CMIP MIB에 대한 Get 명령의 메시지 흐름도



(그림 10) SNMP MIB에 대한 Set 명령의 메시지 흐름도

1. ISODE 통신 연결점(communication point)을 통해서 M_GET PDU를 수신한다.
2. Coordinator와 KS 클래스를 통해서 PDU는 CMISAgent 클래스로 전달된다.
3. CMISAgent 클래스에서 PDU 종류별로 해당되는 멤버함수(CMISAgent :: Get())를 호출한다.
4. 호출된 함수에서는 CORBA의 IDL 인터페이스 또는 RPC의 stub 함수를 이용해서 superMO 클래스에 접근한다.
5. superMO 클래스에서는 접근하려는 관리객체가 공유되고 있는지를 관리객체 참조 테이블인 MO Reference Table를 통해서 확인한다. 만약 동일한 관리객체를 다른 프로세스에서 배타적으로 사용하고 있다면 접근이 보류될 것이다.
6. CMISAgent 클래스에서 호출된 멤버함수는 다시 MO 클래스의 MIB 인터페이스 함수 MO :: Get()에 대응된다.
7. MIB 인터페이스 함수를 통해서 해당되는 관리객체를 탐색 후 그 결과인 관리객체의 값 또는 에러값을 돌려준다().
8. 9. 10. MIB 인터페이스 함수(MO :: Get())에서 돌려 받은 값을 CORBA의 IDL 인터페이스 또는 RPC의 stub 함수를 통해서 CMISAgent의 멤버함수에게 결과 값을 돌려준다.
11. CMISAgent 클래스에서 적절한 PDU를 생성해서 송신한다.

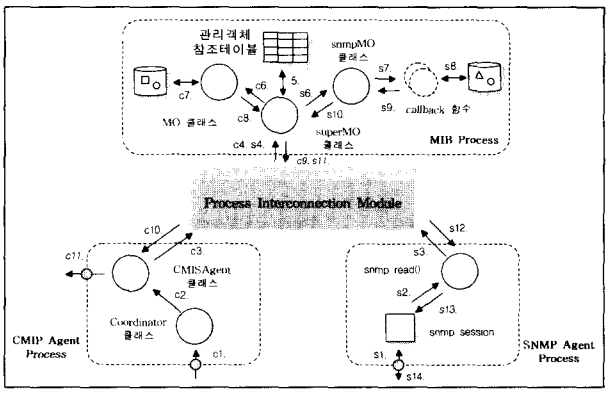
3.3.2 SNMP MIB에 대한 Set 명령

SNMP MIB에 대한 Set 명령의 수행도 위의 Get 명령과 유사한다. 다만 관리객체 참조 테이블(MO Reference Table)에 대한 처리가 다를 뿐이다. 각 단계별 동작모습을 (그림 10)에서 보이고 있다.

1. socket 통신 연결점을 통해서 GET PDU를 수신한다.
2. 각 세션별로 수신된 메시지는 snmp_read() 함수에서 PDU를 분석 한다.
3. 분석한 PDU에서 함수를 호출하기 위한 정보를 추출한다. 그리고 callback 함수를 호출하기 위해서 프로세스 연결 모듈에 접근한다.
4. CORBA의 IDL 인터페이스 또는 RPC의 stub 함수를 이용해서 callback 함수에 대한 함수 포인터를 전달한다.
5. superMO 클래스에서 관리객체 참조 테이블을 참조하여서 동일한 관리객체에 대한 배타적 접근이 가능한지를 확인한다. 접근이 가능하면 접근하려는 관리객체에 대한 정보를 관리객체 참조 테이블에 기록한다.
6. 7. snmpMO 클래스를 통해서 전달 받은 callback 함수를 이용해서 해당되는 관리객체에 접근한다.
8. 관리객체에 대한 명령을 수행한 후 그 결과를 돌려준다. 결과는 callback 함수에 대한 결과 값으로 돌려준다.
9. 10. 11. 12. callback 함수에 대한 결과 값을 분석하여서 해당되는 PDU를 생성한다.
13. 14. socket 통신 연결점을 통해서 생성된 PDU를 송신한다.

3.3.3 공유 MO에 대한 Get 명령

공유되고 있는 관리객체에 대한 두 에이전트의 Get 명령 수행에 대한 동작을 설명하도록 한다. 대부분의 절차는 (그림 9)와 (그림 10)에서 언급한 단계와 유사하다. (그림 11)의 각 단계별 동작모습 중 중복되지 않는 단계만 보였다(그림 11).



(그림 11) 공유 관리객체에 대한 Get 명령의 메시지 흐름도

c1, c2, c3, c4, c6, c7, c8, c9, c10, c11 : (1)의 CMIP MIB에 대한 Get 명령의 각 단계와 동일하다.

s1, s2, s3, s4, s6, s7, s8, s9, s10, s11, s12, s13, s14 : (2)의 SNMP MIB에 대한 Set 명령의 각 단계와 동일하다.

5. superMO 클래스가 관리객체 참조 테이블을 참조하여, 접근하려는 관리객체에 대하여 배타적 접근이 있는지를 확인한다. 만약 배타적 접근이 있다면 현재의 접근은 보류된다. 즉 해당되는 에이전트는 sleep 상태에 들어간다. 배타적 접근이 없어서 현재의 접근이 가능하다면 앞에서 설명한 단계와 동일하게 각각의 MIB 인터페이스 함수(MO::Get(), callback 함수)를 이용해서 해당되는 관리객체에 접근한다.

3.3.4 공유 MO에 대한 Set 명령

마지막으로 공유되고 있는 관리객체에 대한 두 에이전트의 Set 명령의 동작모습을 설명한다. 이 명령의 수행은 앞서의 3가지 경우와는 많이 다른 모습을 보이고 있다. 각 단계별 동작모습을 (그림 12)에서 보이고 있다.

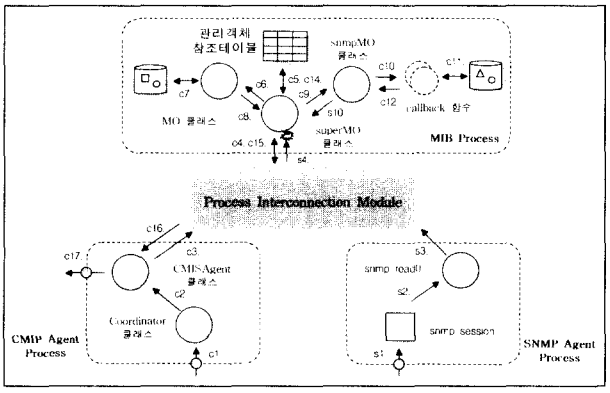
c1, c2, c3, c4, c6, c7, c15, c16, c17 : (3)의 Shared MO에 대한 Get 명령의 각 단계와 동일하다.

s1, s2, s3 : (3)의 Shared MO에 대한 Get 명령의 각 단계와 동일하다.

4. 5. 공유되고 있는 관리객체에 대하여 CMIP 에이전트가 먼저 배타적인 접근(set operation)을 한 이후에 SNMP 에이전트가 동일한 관리객체에 대해서 접근하려고 한다. 나중에 접근하는 SNMP 에이전트의 접근은 보류된다. 즉 상태에 들어간다.

8. 9. 10. 11. 12. 13. CMIP 에이전트의 Set 명령 수행이 끝난 이후 superMO 클래스는 이 공유되고 있는 관리객체에 대한 일관성을 유지하기 위해서 변경된 값을 SNMP측의 관리객체에 대해서 적용한다.

14. superMO 클래스는 CMIP과 SNMP의 MIB에 대한 Set 명령의 수행이 완료된 후 관리객체 참조 테이블(MO Reference Table)을 수정하기 위하여 대기하고 있던 SNMP 에이전트의 접근을 허락한다.



(그림 12) 공유 관리 객체에 대한 Set 명령의 메시지 흐름도

4. 결 론

이상으로 살펴본 바와 같이 본 논문에서는 서로 다른 망 관리 시스템을 통합연동하기 위한 통합 에이전트를 설계 및 구현하였다. 이를 위해서 기존의 방법과는 다른 MOVI (Managed Object View Interface)라는 개념을 이용하였으며, 이 개념은 2장에서도 설명한 바와 같이 기존 방법의 단점이라고 할 수 있는 정보의 손실을 배제할 수 있는 개념이다.

이 MOVI 개념을 지원하기 위해서 논문에서도 제시한 바와 같이 망 관리 시스템으로부터 MIB 부분을 별도의 프로세스로 분리하여 각 에이전트가 동등하게 접근할 수 있도록 하였다. 또한 이렇게 분리된 시스템은 관리객체를 Core 부분과 View 부분으로 분리하자는 초기의 MOVI 개념과 자연스럽게 부합되었다. 그렇게 함으로서 MOVI 개념의 장점인 정보 손실의 배제와 새로운 망 관리 프로토콜의 추가에 대한 확장의 용이성 등의 장점을 갖게 되었다.

이 시스템의 단점이라고 할 수 있는 에이전트 시스템의 복잡함과 구현의 어려움이 있다. 또한 실제 장비에 탑재하기 위해서는 보다 더 작은 용량의 코드와 기존의 알고리즘의 효율화가 필요하다. 프로세스 연결 모듈을 CORBA와 RPC 모두 가능하지만 현재의 구현은 RPC 만을 사용하고 있다. 추후에 CORBA를 이용한 모듈도 테스트 하고자 한다. 그리고 현재 구현은 prototype 형태로 테스트 되었다. 공유 MO에 대한 확장 GDMO 컴파일러에 대한 것은 참고문헌[15]에서 볼 수 있으며 이에 대한 것은 다른 논문에서 구체적으로 언급한다.

참 고 문 헌

[1] Heinz-Gerd Hegering, Sebastian Abeck, Integrated Network and System Management, Addison-Wesley, 1995.

[2] Divakara K.Udupa, Network Management System Essentials, McGraw-Hill, 1996.

[3] 임미경, 김태수, 이광휘, 망관리 프로토콜 연동을 위한 확장된 GDMO , 한국정보처리학회논문집, Vol.7, No.1, pp156-165, Jan. 2000.

[4] Heinz-Gerd Hegering, Sebastian Abeck, Bernhard Neumair, Integrated Management of Networked Systems : Concepts, Architectures, and Their Operational Application, Morgan Kaufmann Publishers, Inc., 1999.

[5] K.-H. Lee, MOVI : Management Object View Interface for Hierarchical Distributed Network Management System, pp.12-16, IEEE ICC Volume 1, 1996.

[6] CMU, UCD, "The NET-SNMP Project Home Page," http : //net-snmp.sourceforge.net/, October, 2000.

[7] G. Pavlou, G. Knight, "The OSI Mangement Information Service User's Manual, Version 3.0," January, 1993.

[8] 박상철, 김태수, 이광휘, "MOVI 개념을 이용한 통합 Agent 구현 모델", Technical Report, TR00-CSL01, 창원대학교 전자계산학과, 2000.
[9] William Stallings, SNMP, SNMPv2, and CMIP : The Practical Guide to Network-Management Standards, Addison-Wesley, 1993.
[10] David Perkins, Evan McGinnis, Understanding SNMP MIBs, Prentice Hall, 1997.
[11] Keith Haviland, Dina Gray, Ben Salama, Unix System Programming : A programmer's guide to software development, Addison-Wesley, 1995.
[12] Microsoft, MSDN Library Visual Studio 6.0, 1999.
[13] Jon Siegel, CORBA Fundamentals and Programming, John Wiley & Sons, 1996.
[14] W.Richard Stevens, Unix Network Programming, Prentice Hall, 1994.
[15] 김종만, 통합 망관리 시스템 지원을 위한 Managed Object Definition에 관한 연구, 창원대학교 석사학위논문, Dec. 2000.

박 상 철

e-mail : sunny@cdcs.changwon.ac.kr
1993년 창원대학교 자연과학대학 전자계산학과(학사)
2001년 창원대학교 대학원 컴퓨터공학과(이학석사)
2001년~현재 (주)핸디소프트
관심분야 : 통신망관리, 멀티캐스트 프로토콜, 이동통신망

김 태 수

e-mail : jupi@sarim.changwon.ac.kr
1995년 경상대학교 농과대학 임산공학과(학사)
1998년 창원대학교 대학원 컴퓨터공학과(이학석사)
1999년~현재 창원대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 통신망관리, 멀티캐스트 프로토콜, 이동통신망, 시뮬레이션

이 광 휘

e-mail : khlee@sarim.changwon.ac.kr
1983년 고려대학교 공과대학 전자공학과 졸업(학사)
1985년 고려대학교 대학원 전자공학과 통신전공(공학석사)
1989년 고려대학교 대학원 전자공학과 컴퓨터전공(공학박사)
1988년~현재 창원대학교 컴퓨터공학과 교수
1991년~1993년 영국 Walse대학(Swansea) 및 Reading대학교 연구원
1995년~1996년 영국 런던대학(UCL) 연구원
1997년~1998년 영국 Walse대학교(Swansea) 및 New Bridge Networks사 연구원
2000년~현재 영국 Walse 대학교(Swansea) 연구원
관심분야 : 망관리시스템, 멀티캐스트 프로토콜, 이동통신망, 분산시스템, QoS Management