객체 지향 개념을 적용한 성능 모니터의 구현

김 용 수[†] · 이 금 석[††]

요 약

프로세서의 속도, 주기억장치의 크기 및 엑세스 속도, 입출력 대역폭 등 컴퓨터의 물리적 속성은 사용자에게 주어질 때 고정되어 있다. 이러한 제한 하에서, 여러 사용자가 컴퓨터의 자원을 공유하는 다중 프로세스 시스템의 성능은 사용자 프로세스와 자원의 상관 관계를 모니터하고 조정함으로써 향상될 수 있다. 본 논문은 객체 지향 개념을 성능 관리에 적용하여 객체화된 시스템의 자원 관리자와 사용자 프로세스 및 성능 관리자 사이의 대화 프로토콜 및 객체의 속성을 정의함으로써 성능 관리 시스템의 표준을 제시한다. 이러한 표준화를 통해 성능 관리의 대상이 되는 객체와 성능 관리자를 독립적으로 개발할 수 있고, 성능 관리자를 통해 시스템의 성능을 통합적으로 관리할 수 있다.

Implementation of a Performance Monitor using Object Oriented Concept

Yong Soo Kim[†] · Keum Suk Lee[††]

ABSTRACT

The physical attributes of a computer, such as processor speed, size and access time of memory, and I/O bandwidth, are fixed when the computer is delivered to the user. Under these conditions, the performance of a multi-process system where multiple processes share various resources can be enhanced by monitoring and controlling the relationship between the user processes and the system resources. This paper applies object oriented concept to the performance management and suggests a standard for the management. Resource managers and user processes are defined as managed objects and a performance manager is defined as a managing object. A protocol between the user processes and the performance manager is designed and attributes and methods of the objects are also defined. Through the standardization, the user processes and the performance manager can be developed independently and the system's performance can also be collectively managed.

1. 서 론

컴퓨터 시스템의 성능 목표는 사용자의 입장에서 작업 처리율을 최대화하고 응답시간을 최소화하는 것이다. 사용자에게 주어진 컴퓨터 자원들, 예를 들어 CPU, 주기억장치, 입출력 장치들은 그 크기나 성능이 제한되어 있다. 또 다중 프로세스 시스템에서 작업 처리율을 올리기 위해서는 이들 자원들의 활용도를 높혀야하나 이 경우 사용자가 자원을 사용하기 위해 대기하는 시간이 길어지므로 응답시간이 길어진다. 그러므로 시스템의 성능을 향상시키기 위해서는 사용자, 자원 그리고 사용자와 자원의 관계를 파악하

† 정 회 원:현대정보기술주식회사
†† 정 회 원:동국대학교 컴퓨터공학과
논문접수:1997년 3월 4일, 심사완료:1997년 5월 15일

는 것이 중요하다.

수행중인 시스템의 성능을 온라인으로 모니터할 수 있다면 성능을 저해하는 병목현상을 찾아냄으로써 자원 또는 사용자 프로세스의 분산, 재배치 등을 통해 성능을 증대할 수 있다[1]. 일반적으로 성능 모니터는 모니터 대상의 작업 과정에 관한 정보를 수집하며[2], 대상 외부에 일반적으로 영구히 존재한다[3]. 성능 관리는 모니터된 성능 데이터를 평가하는 부분과 그에 따라 성능을 향상시키는 부분으로 나눌 수 있고 향상 부분은 시스템 자원의 동적 분배, 프로그램의 동적 조율, 온라인 프로그램 수행조정(steering) [4] 등을 수행한다. 종래의 모니터는 자원의 활용도를 중점적으로 다루든지 또는 응용 프로그램의 성능을 중점적으로 다루었고 또 특정 시스템에 국한된 것이었다. 그러나 시스템의 성능은 사용자와 자원의 상관관계로 파악되므로 양자를 종합적으로 파악하는 것이 중요하며 또 성능 관리를 표준화함으로써 이식성(portability)과 상호작용성(interoperability)을 높일 수 있다.

모니터는 사건중심, 추적중심 및 샘플중심의 모니터로 나눌 수 있는데 본 논문은 Schroeder[3]가 제시한 세 가지 사건중심의 유형중, 프로세스-레벨과 응용 프로그램 종속-레벨에 대해 다루고자 하며 모니터에 전달되는 사건의 수를 줄이기 위해 한계값[5]도 프로토콜에 포함하였다. 감지기(sensor)는 대상 내에 존재하면서 대상의 상태 변화에 따라 혹은 모니터의 요청에 의해 사건을 발생시키는 것[6]으로 모니터되는 대상과 동기적으로 작동한다. 이러한 감지기는 흔히 계측에 의해 만들어진다.

컴퓨터 시스템의 성능에 영향을 주는 요소들로는 하드웨어, 운영체제 및 응용 프로그램 등이 있다[7]. 운영체제는 시스템의 자원을 관리하고 사용자에게 분배하는 역할을 하므로 자원의 성능은 운영체제를 모니터하여 얻을 수 있다. 본 논문은 컴퓨터 시스템 자원의 모니터링에 대한 객체 지향 개념의 설계[8]를 사용자, 즉 응용 프로그램 중심으로 확장한 것으로서, 컴퓨터 시스템에는 각 자원을 관리하는 자원 관리자와 그 자원들을 사용하는 사용자들이 있고, 자원 관리자와 사용자가 성능 관리에 능동적으로 참여함을 가정하였다. 즉 측정 대상이 되는 자원 관리자나 사용자가 자신의 성능을 측정하고 이를 성능 관리자에게 전달하는 방법이다. 여기서는 성능 측정을 위한 온라인 모니터를 중심으로 성능 관리에 참여하는 요소들을 객체로 정의하고, 성능 데이터를 속성으로 가지는 자원 관리자 및 사용자와 성능 관리자간의 대화 프로토콜을 정의하고 설계하였다.

일반적으로 사용자는 컴퓨터 시스템에게 작업지시를 하는 사람 또는 그 사람이 사용하는 명령이나 트랜잭션, 사용자 프로세스 또는 응용 프로그램을 수행하는 프로세스 무리를 이루는 작업 등 여러 의미로 사용될 수 있다. 본 논문에서는 사용자의 의미를 응용 프로그램 또는 프로세스로 사용하였다.

본 논문의 구성은 7장으로 되어 있다. 2장은 기존 시스템에 대해 살펴본 것이다. 3장은 시스템의 구성에 대한 것이며 4장에서는 본 논문이 제시한 성능 관리 시스템의 객체 지향적 요소들에 대한 논의이다. 5장은 성능 관리 객체간의 대화 프로토콜의 정의이며, 6장은 시스템의 구현 및 고려사항을 다루었다. 7장에서는 결론 및 앞으로의 연구 방향에 대해 언급하였다.

2. 기존 시스템의 고찰

자원의 활용도를 모니터하는 시스템[9, 10, 11, 12]은 많이 개발되어 있고 이를 객체 지향적으로 구성하려는 시도도 있다[8, 13, 14, 15]. 응용 프로그램의 성능 측정은 주로 계측을 통해 이루어지고 있으며 정적인 것과 동적인 것으로 나눌 수 있다. 정적인 계측은 계측 코드를 프로그램에 삽입하여 컴파일하고, 수행중에 수집된 데이터는 파일에 저장된 후 나중에 분석되는 것으로 대부분의 프로그램 계측은 여기에 속한다[16, 17]. 동적 계측은 계측 코드를 수행중인 프로그램에 삽입하여 성능 데이터를 얻는 것으로 qpt[17], Pixie[18], OMIS[19], Paradyn[20] 등이 여기에 속한다. OMIS는 분산환경을 모니터하는 다양한 도구들간의 인터페이스를 표준화함으로써 이식성과 상호작용성을 높이는 목적으로 개발되었다. Paradyn은 시스템의 병목현상을 "왜", "어디서", "언제"의 세 개의 변수로 파악하는 W3 탐색모델[21]을 적용하여 온라인으로 성능 데이터를 수집하고 이 데이터를 시간 히스토그램[22]이라는 데이터 구조에 저장한다. Paradyn의 또 다른 특징으로는 성능에 문제가 있는 부분만 동적으로 계측할 수 있는데 있다[23]. 그러나 동적 계측은 기

계어 레벨에서 일어나야 하므로 시스템에 종속적이며 고급언어의 문장을 계측하기 어렵다. 실제로 Paradyn에서는 기계어 레벨에서, 계측점을 찾기 쉬운 프로시쥬어 진입점과 출구 및 함수 호출에만 계측을 하고 있다. 성능 측정의 부하를 줄이기 위해서 ISSOS[24]와 EDL[25]과 같이 컴파일 시에 프레디키트(predicate)를 넣어서 성능 데이터를 선별적으로 수집하는 방법과 BEE[26]와 Paradyn과 같이 동적으로 하는 방법이 있다.

분산시스템에서의 성능 관리의 문제점으로는, 1) 데이터의 전송지연으로 인한 현시점의 종합된 정보가 얻기 어렵다, 2)전송지연이 일정치 않아서 사건간의 전후 뒤바뀜, 3)과다한 사건발생으로 부하, 4)이기종 간의 성능 데이터와 데이터의 전송에 관한 표준 프로토콜 필요 등을 들 수 있다. 분산환경에서 사건중심과 샘플중심의 모니터를 지원하고 사용자가 모니터될 사건을 정의하며 온라인으로 프로그램 수행을 조정할 수 있는 시스템으로 Falcon[27]을 들 수 있고 Borgeest는 분산환경에서 표준 접속을 시도하였다[19].

3. 성능 관리 시스템의 구성

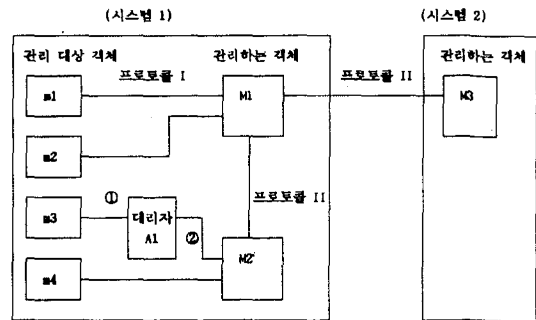
3.1 구성요소

성능 측정의 대상이 되는 것은 자원과 사용자 프로세스 그리고 자원과 사용자 프로세스간의 관계이다. 시스템의 자원은 자원 관리자에 의해 사용자에게 할당된다. 시스템의 기본 자원으로는 프로세서, 기억장치, 입출력 장치 등의 물리적 자원이 있고 IPC, 각종 버퍼나 큐, 파일 시스템, 가상 기억장치 등의 기본 자원으로부터 파생된 논리적 자원이 있다. 시스템의 자원은 해당 관리자에 의해 관리된다고 가정하였으므로 앞으로 특별히 구별할 이유가 있을 때를 제외하고는 자원을 자원 관리자의 의미로 사용한다. 이러한 자원 관리자는 관리 대상 객체로 정의되고 성능 메트릭을 속성으로 가지며 속성은 자신의 메소드에 의해서만 조작된다. 자원 관리자나 사용자의 성능 데이터는 성능 관리자에게 프로토콜에 의해 전달되는데 이 성능 관리자는 관리하는 객체로 정의되고 데이터의 분석, 가공, 시스템의 제어, 사용자 인터페이스에 데이터를 전달하는 일 등을 하며 자원 관리자와 같은 시스템에 존재한다.

운영체제가 직접 관리하는 자원뿐만 아니라 서브시스템 및 서브시스템의 각 구성 요소(논리적 자원 관리자)도 관리 대상 객체가 될 수 있다. 이 경우 성능 관리자는 운영체제 및 그 서브시스템의 통합된 성능 정보를 동시에 가지고, 그 상관 관계를 분석 할 수 있다. 자원 관리자는 자신이 가진 성능 메트릭의 프로토타입을 성능 관리자에 등록하고 성능 메트릭을 전달함으로써 능동적으로 성능 관리에 참여한다. 이렇게 함으로써 자원 관리자와 성능 관리자의 독립적인 개발이 가능하다. 대리자는 자원의 성능 데이터를 자원 관리자의 도움 없이 수집하는 객체로서 자원의 데이터를 읽거나 샘플중심의 모니터를 시행한다. 성능 관리자는 대리자를 관리 대상 객체로 취급한다. 자원 관리자는 시스템 내에 영속적으로 존재하나 사용자는 필요에 따라 생겨나거나 없어질 수 있다.

3.2 구성도

(그림 1)은 앞에서 설명한 객체간의 관계를 나타낸 것이다. m1, m2, m3, m4는 관리 대상 객체로서 자원 또는 사용자를 나타낸다. M1, M2, M3은 관리하는 객체로서 성능 관리자를 나타내고 A1은 대리자를 나타낸다. 프로토콜 I은 관리 대상 객체와 관리하는 객체 사이의 대화 규칙을 정한 것으로 성능 메트릭의 등록 및 삭제, 성능 데이터의 수집 및 제어 명령의 전달 등을 규정하고 있다. 프로토콜 II는 관리하는 객체 사이의 대화 규칙을 정한 것으로 부하의 분배 등을 규정하고 있으며 분산 환경의 성능 관리에 확장되어 사용된다. ①은 대리자가 임의로 자원으로부터 데이



(그림 1) 시스템 구성
(Fig. 1) System configuration

터를 수집하는 것으로 정해진 규칙이 없고, 관리하는 객체의 입장에서 대리자는 관리 대상 객체이므로 ②는 프로토콜 I과 같다.

(그림 1)과 같은 구조는 다음 몇 가지의 장점을 가지고 있다. 첫째, 시스템의 성능은 여러 자원과 사용자의 관계에서 파악될 수 있으므로, 관리 대상 객체간에 성능 조율을 위한 프로토콜이 있는 것이 이상적이다. 그러나 관리 대상 객체간의 모든 관계를 미리 아는 것이 어렵고, 또 자원 관리자는 운영체제의 일부분으로서 변경하기가 용이하지 않다. 그러나 (그림 1)에서, 관리하는 객체는 사용자 주소 공간에 존재하고 시스템의 모든 성능 데이터를 모아서 통합적으로 관리함으로써 변경이 용이하고, 관리 대상 객체간의 간접적인 대화 통로가 될 수 있다. 둘째, 관리 대상 객체와 관리하는 객체가 한 시스템 내에 존재하므로 많은 성능 데이터를 처리하기에 유리하다. 즉 이진(binary) 데이터 및 식별자가 관리 대상 객체와 관리하는 객체 사이에 전달되고 또 공유 기억장치 등이 사용 될 수도 있다. 셋째, 관리하는 객체는 통합된 성능 자료를 가지고 있으므로 성능 관리 전문가 시스템으로 발전시킬 수 있다. 넷째, 시스템의 관리하는 객체간의 프로토콜을 정의함으로 분산 환경에서의 성능 관리 시스템을 구축할 수 있다.

4. 성능 관리 시스템의 클래스와 객체

본 논문은 성능 관리 시스템의 다음과 같은 부문을 객체 지향으로 설계하였다.

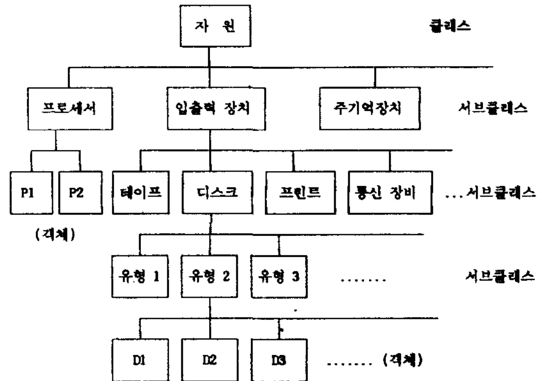
1) 자원 관리자와 사용자는 객체이다.

2) 성능 관리자는 객체이며 자원 관리자 및 사용자와 프로토콜 메시지를 통해 모니터와 제어가 이루어진다.

3) 성능 메트릭도 분류되어 클래스로 정의된다.

4.1 물리적 자원

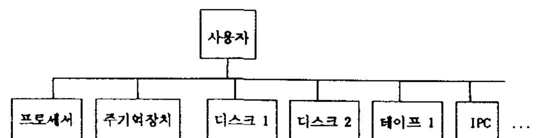
객체로 정의된 시스템의 물리적 자원 클래스의 계층 구조는 (그림 2)와 같이 나타낼 수 있다. 프로세서 클래스는 다중 프로세서의 경우 프로세서 수만큼 객체로 실체화되어 자신에게 일어난 인터럽트 수와 같은 객체의 속성을 가질 수 있다. 입출력 장치는 장치의 종류, 장치의 유형 등의 서브클래스를 가진다. 각 계층의 클래스에서는 자신에 합당한 속성과 메소드가 정의된다. (그림 2)에서 디스크 유형 2의 객체들, D1, D2, D3... 등은 해당 디스크의 성능 메트릭을 속성으로 가진다.



(그림 2) 자원의 클래스 계층 구조의 예
(Fig. 2) An example of the class hierarchy of the resource

4.2 사용자

일반적으로 사용자의 응답 시간은 사용되는 자원들의 서비스 시간으로 세분되어 분석된다. 사용자 또는 자원 위주의 세분을 할 수 있으며 (그림 3)은 사용자를 위주로 세분한 것이고, Berry와 Maccabee [13]는 자원 위주의 모델을 제시하고 있다. (그림 3)에서 세분된 자원은 사용자의 해당 자원의 사용에 대한 성능 데이터를 가지고 있다. 자원은 자원 관리자에 의해 관리되므로 자원 위주 모델은 구현하기 쉬우나, 사용자가 자신의 성능 데이터를 모니터하여 (그림 3)과 같은 데이터 구조를 갖기는 어렵다. 사용자 프로그램의 계측을 통해 사용자의 DBMS 자원 사용을 모니터한 것[16]은 사용자 위주로 구현된 성능 모니터의 예이다.



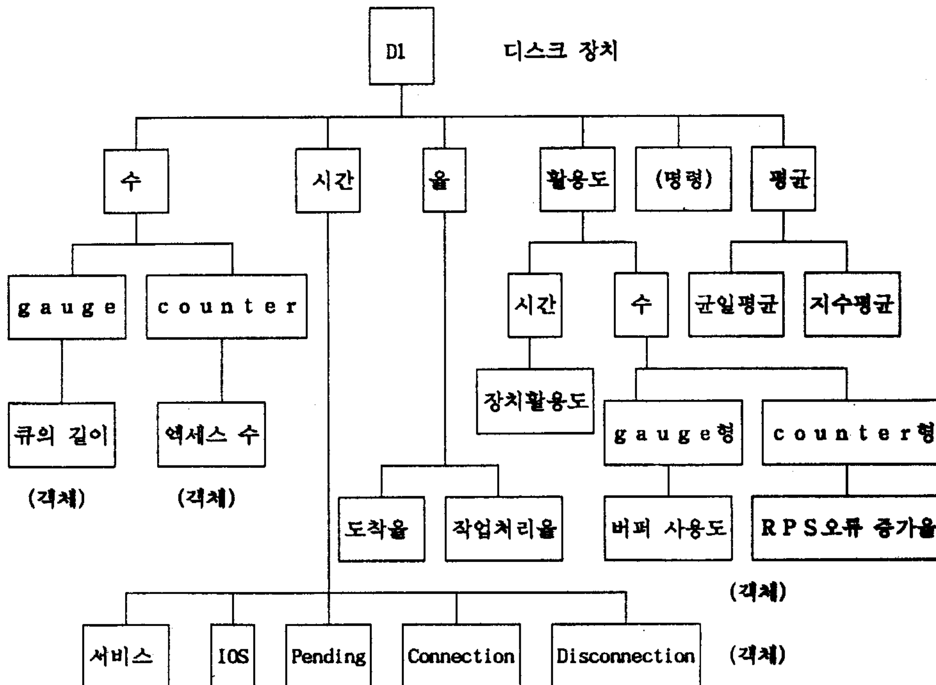
(그림 3) 사용자의 자원 세분
(Fig. 3) Resource usage of a user

4.3 객체의 성능 데이터

시스템의 성능을 나타내는 데이터로는 응답시간, 작업 반환시간, 작업 처리율, 자원의 활용도, 작업이나 트랜잭션의 도착율 및 상주 시간(residence time), 큐의 길이, 지연 시간, 자원의 사용 빈도 등이 있다. 또 터미널 사용자의 관점에서 사용자의 명령을 수행하는 속도를 나타내는 성능값으로는 수행시간, 처리율 및 적시성(timelyness) 등이 있다[28]. 이러한 성능 데이터는 수(number)와 시간(duration)을 측정하여 율(rate)이나 활용도(utilization), 평균값 등으로 추출된 것이다. Berry[13]는 성능 데이터를 시간, 수, 율로 분류하였고 본 논문에서는 수를 ISO의 정의[29]에 따라 증감하고 음이 아닌 정수를 나타내는 게이지(gauge)와 증가만 하는 음이 아닌 정수인 계수(counter)로 세분하고, Berry의 정의에 활용도 및 평균을 첨가하여 성능 메트릭 클래스로 정의하였다.

(그림 4)는 (그림 2)의 디스크의 객체인 D1의 성능 메트릭을 클래스 구조로 나타낸 것이다. 성능 메트릭 클래스에는 각 메트릭을 조작하는 매소드가 정의된다. 성능 관리의 대상이 되는 객체로서의 자원은 자신의 속성인 성능 데이터를 (그림 4)와 같이 객체 지향적으로 표현한다. 성능 매트릭 클래스 계층에 표시된 명령은 D1을 제어하기 위한 클래스로서 모니터할 성능 메트릭과는 관계가 없다. 균일 평균은 모든 샘플의 비중을 같다고 가정하여 계산한 평균이며 지수 평균은 가장 최근의 값에 비중을 두어서 계산한 평균이다.

(그림 2)와 (그림 4)에 의거한 기본 자원의 성능 데이터에는 다음과 같은 것들이 있다. 스케줄러는 프로세서 자원 관리자의 한 부분이므로, 게이지 클래스에는 준비(ready) 큐 길이, 대기(wait) 큐 길이, 계수 클래스에는 인터럽트 수 등이 포함되고, "시간" 클래스에는 대기 시간, 서비스 시간 등이 있으며, "활용도" 하의 "시간" 클래스에는 장치의 활용도, "율" 클래스에는 도착율 등이 있다. 주기억장치 성능 데이터의 게이지 클래스에는 활용 가능 프레임 수, 사용중인



(그림 4) 디스크의 성능 메트릭 클래스 구조의 예
(Fig. 4) An example of the class hierarchy of the disk performance metrics

프레임 수 등이 있다. 디스크의 게이지 클래스에는 큐의 길이, 계수 클래스에는 엑세스 횟수, "시간" 클래스에는 입출력 장치 큐에서 대기하는 시간(IOS), 입출력 명령이 시행된 시간부터 실제 장치가 가동을 시작할 때까지의 시간(pending), 장치가 입출력 채널과 연결되어 있는 시간(connection), 장치가 탐색 또는 섹터 설정 명령을 받아서 채널과 연결되지 않고 작동하는 시간(disconnection), 그리고 앞서 말한 모든 시간을 합한 장치의 서비스 시간 등이 있다.

(그림 4)의 각 성능 매트릭 클래스의 실체화인 객체는 〈표 1〉, 〈표 2〉, 〈표 3〉, 〈표 4〉와 같은 성능 데이터를 속성으로 가지며, 관리 대상 객체는 〈표 6〉의 정적인 데이터 및 〈표 7〉의 제어 데이터를 속성으로 갖는다.

4.3.1 성능 데이터

수와 시간은 기본 성능 메트릭이고 평균값, 율, 활용도 등은 이를 기초로 얻을 수 있으며 이를 세 가지의 유형으로 나눌 수 있다. 〈표 1〉은 세 가지 유형에서 공통적으로 사용되는 속성들이며 〈표 2〉, 〈표 3〉, 〈표 4〉는 각각의 유형을 기술하고 있다. 또 이들 표에서 유형 필드는 ISO와 CCITT의 정의를 사용했고, 부동소수(Float)와 플래그(Flag)를 첨가했다. 보호 필드는 성능 관리자가 관리 대상 객체의 해당 속성을 설정하지 못하게 보호하려는 것으로, "Y"인 경우 성능 관리자가 값을 설정할 수 없다. 객체 식별자는 성능 관리자에 의해 부여되고 시스템 내에서 유일하며, 데이터 식별자는 관리 대상 객체가 부여하고 객체 내에서 유일하다. 현재시간은 성능 데이터가 가장 최근에 수집된 시간이다. 기준시간은 평균, 율, 활용도 등을 계산하는 기준이 되는 시간이며 빈돗수는 이들 값이 샘플링된 횟수를 의미한다. 보고상태는 값이 한계값을 초과할 경우나 한계값을 초과한 후 다시 정상한계값 이하로 떨어진 경우(eventReport)와 값에 변화가 있을 경우(statusReport)에 트랩 메시지를 성능 관리자에게 보낼 것인지를 결정한다. EventReport와 status Report의 설정은 관리하는 객체가 메시지를 통해 관리 대상 객체에 요청한다. 한계_1 필드에는 관리 대상 객체가 정한 한계값이 들어가며, 값이 이 한계값을 초과하고 eventReport가 설정되어있으면 관리 대상 객체는 심각한 성능 저하를 경고하는 트랩 메시지를 성능 관리자에게 보낸다. 한계_2 필드에는 성능 관리자가 정한 한계값이 들어가며, 값이 이 한계값을 초과하고 eventReport가 설정되어있으면 역시 트랩 메시지가 전송된다. 성능 관리자가 설정하지 않으면 한계_2 값은 한계_1 값과 같다. 자원의 성능값이 한계값 주변에서 변화가 심할 경우 많은 사건이 일어남으로 정상한계값 이하로 떨어질 때 성능이 정상임을 관리하는 객체에게 보고하게 한다. 또 〈표 1〉, 〈표 2〉, 〈표 3〉, 〈표 4〉의 데이터를 수집 주기마다 이력(history) 파일에 기록해 둠으로써 추후 시스템 성능의 경향 등 각종 정보를 추출할 수 있다

〈표 1〉 성능 데이터의 공통 속성
〈Table 1〉 Common attributes of the performance data

속성	유형	보호	설명
식별자	Integer	Y	데이터의 식별자
이름	DisplayString	Y	데이터의 이름
다음_포인터	Integer	Y	다음 속성을 가리키는 포인터
유형	Flag	Y	본 모델이 정한 '값'의 속성 유형(X'01')
값	Integer	N	유형에 준한 현재의 값
현재시간	TimeTicks	Y	값을 모니터한 시간
기준시간	TimeTicks	Y	평균, 율, 활용도의 기준 시간
빈돗수	Integer	Y	샘플 횟수
한계_1	Integer	Y	사건(event)을 유발하는 한계값
한계_2	Integer	N	사건을 유발하는 한계값
정상한계	Integer	N	성능이 정상임을 알리는 한계값
보고상태	Flag	N	사건발생과 상태변화의 전송을 표시

〈표 2〉는 첫 번째 유형으로서 〈표 1〉의 속성들을 계승한다. 속성인 값은 현재의 큐의 길이, 버퍼에 남아있는 블록의 수, 프로그램 블록의 수행시간 등, 증감하고 음이 아닌 정수를 나타내는 게이지이다. 샘플은 큐의 길이가 바뀔 때마다 일어난다. 이 유형에서는 값으로부터 최대값, 최소값, 평균값 등의 성능 메트릭이 추출된다.

〈표 1〉의 속성들을 계승한 두 번째 유형은 〈표 3〉과 같이 표현되며 값은 디스크의 엑세스 횟수, 트랜잭션의 도착수 등, 증가만 하는 음이 아닌 정수인 계수이다. 샘플은 값이 증가할 때마다 일어난다. 이 유형에서는 측정된 계수의 차이인 게이지와 시간을 사

〈표 2〉 성능 데이터의 속성 유형 1
〈Table 2〉 Type 1 attributes of the performance data

속성	유형	보호	설명
〈표 1〉	〈표 1〉의 속성 계승		
최소값	Integer	Y	유형에 준한 최소 값
최대값	Integer	Y	유형에 준한 최대 값
평균값	Float	Y	유형에 준한 평균 값

〈표 3〉 성능 데이터의 속성 유형 2
〈Table 3〉 Type 2 attributes of the performance data

속성	유형	보호	설명
〈표 1〉	〈표 1〉의 속성 계승		
기준횟수	Integer	Y	율을 계산하기 위한 기준횟수

용해서 율을 구한다.

〈표 4〉는 역시 〈표 1〉의 성능 속성을 계승한 세 번째 유형으로서 시간을 시간으로 나누거나, 게이지(또는 계수를 게이지로 만든 후)를 같은 성격의 수로 나누어서 활용도, 비율, 증감율 등을 추출한다. 전체 모니터 시간에 대한 디스크의 사용시간은 활용도에 속하며, 일정 시간 동안 계수를 모니터해서 그 증감율을 구할 수 있고, 게이지에 대해서는 자원이 수용할 수 있는 최대값인 극한값과의 비율을 구할 수 있다. 샘플은 대기 상태의 변화 등에서 일어난다.

〈표 4〉 성능 데이터의 속성 유형 3
〈Table 4〉 Type 3 attributes of the performance data

속성	유형	보호	설명
〈표 1〉	〈표 1〉의 속성 계승		
축적값	Integer	Y	활용도의 계산에 사용된다
기준값	Integer	Y	계수의 증감율 계산에 사용된다
극한값	Integer	Y	게이지의 비율 계산에 사용된다

〈표 5〉 유형의 특성
〈Table 5〉 Characteristics of the attribute types

구분	유형 1		유형 2	유형 3	
기본 속성	수	시간	수	시간	수
나누는 수	빈돗수		시간	시간	수
추출된 속성	평균, 최대, 최소		율	활동율(%)	

이들 세 가지 유형의 특성은 다음 〈표 5〉로 요약된다. 또 이들 유형은 객체 지향 언어로 쉽게 표현할 수 있다.

4.3.2 정적인 데이터

〈표 6〉의 정적인 데이터는 객체의 전체적인 상태를 나타내는 데이터이다. 활용도, 평균 응답시간 등 많은 성능 메트릭은 일정 기간 성능 데이터를 수집한 후 계산을 해서 얻는 값이다. "수집주기" 속성은 이 수집 시간을 나타내는 것으로 성능 관리자가 설정할 수 있다. "객체상태"는 관리 대상 객체의 현재 상태를 나타내는데, 객체가 성능 데이터를 수집하고 있을 때(active), 주기적으로 관리하는 객체에게 데이터를 전송하고 있을 때(autoReply), 성능 데이터가 한계값보다 클 때(threshold_1, threshold_2), 성능 데이터의 수집을 잠정적으로 중단한 때(inactiveTemp) 또는 영구적으로 중단한 때(inactivePerm) 등으로 정의된다. 객체의 상태가 inactiveTemp 또는 inactivePerm이면 관리 대상 객체는 성능 데이터의 수집을 중지하고 성능 관리에 참여하지 않는 상태이며, 그 이유는 "다운_이유"에 기록된다. "다운_이유"에는 관리 대상 객체가 성능 데이터 수집의 중단을 선언했을 경우(deleteManagedObject), 관리하는 객체가 선언했을 경우(deleteManagingObject) 등 누가 중단을 요청했느냐에 대한 것과, 객체 사이의 과다한 통신 부하(overheadMessage), 성능 데이터의 수집 부하(overheadCollection), 프로토콜의 오류(protocolError) 등 상세 내용으로 정의된다.

〈표 6〉 정적인 데이터
〈Table 6〉 Static data

속성	유형	보호	설명
시작시간	TimeTicks	Y	성능 관리자에게 클래스를 등록한 시간
수집주기	TimeTicks	N	데이터의 수집 시간 주기
다음_포인터	Integer	Y	다음 성능 데이터를 가리킨다
객체상태	Flag	Y	객체의 성능 모니터 수집 상태
다운_이유	Flag	Y	객체의 성능 관리 기능이 멈춘 이유

4.3.3 제어 데이터

〈표 7〉은 객체가 갖는 제어 속성을 나타낸 것이다.

자원 관리자가 명령을 성능 관리자에게 등록하면, 성능 관리자는 필요시 명령의 식별자를 이용하여 제어 명령을 자원 관리자에게 보낸다. 매개변수에는 명령의 수행을 다양하게 하는 명령의 매개변수가 들어간다.

〈표 7〉 제어 데이터
〈Table 7〉 Control data

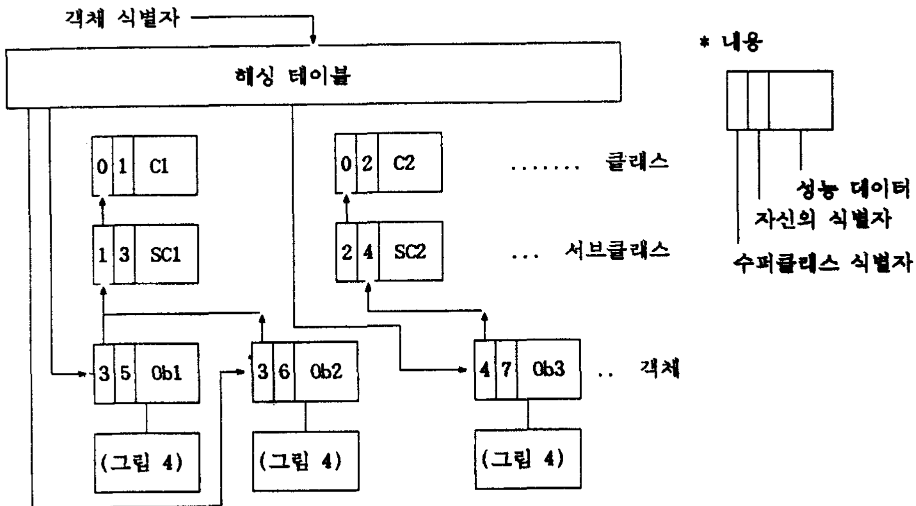
속성	유형	보호	설명
식별자	Integer	Y	명령의 식별자
설명	DisplayString	Y	명령 및 매개변수의 설명
매개변수	[Integer]	Y	명령의 매개변수, [ ]는 형태의 반복 정의

4.4 객체의 메소드

객체의 메소드는 CREATE, DELETE, GET, SET, ACTION, CANCEL, EVENT-REPORT, GET-NEXT, AUTO-REPLY, STATUS-REPORT, MIGRATE, INFORM 등 12개로 정의된다. 관리하는 객체의 CREATE 메소드는 관리 대상 객체로부터 성능 데이터의 프로토타입을 받아서 객체화하여 성능 관리 데이터베이스를 만든다. DELETE는 관리하는 객체와 관리 대상 객체가 가지는 메소드로서 한 객체로부터 성능 관리의 중지를 요청하는 메시지를 받아서 객체간의 대화 채널을 삭제한다. EVENT-REPORT는 관리하는 객체의 메소드로서 관리 대상 객체가 한계값을 넘는 성능 데이터가 모니터될 때 보내는 트랩 메시지를 처리한다. GET와 GET-NEXT는 관리 대상 객체의 메소드로서 관리하는 객체로부터 성능 데이터 요구를 받아서 그 응답을 관리하는 객체에게 준다. 관리 대상 객체의 메소드 SET는 관리하는 객체로부터 성능 데이터 값의 설정 요구를 받아서 처리한다. ACTION은 관리하는 객체로부터의 명령을 수행하는 관리 대상 객체의 메소드이다. 관리 대상 객체의 CANCEL은 지금 수행 중인 요청을 취소하며, AUTO-REPLY는 주기적으로 자신의 성능 데이터를 모니터하여 관리하는 객체에게 전달하는 메소드이다. STATUS-REPORT는 관리하는 객체의 메소드로서 성능 데이터의 값이 변할 때 자발적으로 전송된 메시지를 처리한다. MIGRATE는 관리하는 객체 상호간에 관리 대상 객체를 나누는 메소드로서 부하의 균형을 위한 것이다. INFORM은 관리하는 객체 상호간의 정보를 교환하는 일을 담당하는데 분산 환경에서의 성능 관리에 확장되어 사용된다.

4.5 성능 관리자의 데이터베이스 모델

성능 관리자는 자원 관리자로부터 성능 데이터의 프로토타입을 받아서 동적으로 데이터 클래스를 실



(그림 5) 성능 관리자의 데이터베이스 모델
(Fig. 5) Database Model of the performance manager

체화하여 데이터베이스를 구축한다. 그러므로 클래스와 객체의 계층적 구조를 (그림 5)와 같이 동적으로 구성해야 한다. (그림 5)에서 C1, C2는 클래스, SC1, SC2는 서브클래스, Ob1, Ob2, Ob3은 객체를 나타내고 각 객체는 (그림 4)와 같은 성능 메트릭 구조를 가진다. 객체 식별자는 키(key)가 되어 해싱 테이블(hashing table)을 통해 해당 객체를 찾아가며 그림에서 해싱으로 생기는 동의어의 표시는 생략되었다. 객체는 클래스를 가리키는 포인터를 가지고 있으므로 클래스의 메소드와 속성을 엑세스할 수 있다. 같은 이유로 서브클래스는 슈퍼클래스를 가리키는 포인터를 가진다. 클래스 및 서브클래스와 객체 사이의 관계 표시는 객체 지향의 고유한 것이고 자원 관리자도 이와 같은 데이터 구조를 가진다. 클래스와 객체 데이터는 크게 세 가지 필드로 구분할 수 있는데 첫 필드는 슈퍼클래스의 포인터, 둘째 필드는 자신의 식별자, 객체의 세 번째는 4.3에서 정의된 성능 데이터가 들어간다.

클래스 데이터, 즉 C1, C2, SC1, SC2 등의 세 번째의 성능 데이터 부분에는 〈표 8〉과 같이 실체화된 객체들 중에 한계값을 초과한 비율이 가장 높은 객체와 데이터의 식별자 및 한계값을 초과한 객체와 데이터의 수를 보관한다. 이러한 구조는 성능 관리자가 모든 객체를 모니터하지 않고 전체적인 성능을 파악할 수 있게 한다. 여기서 성능 관리자가 설정한 한계값을 초과한 비율을 한계비율이라고 하고 다음과 같이 정의한다.

$$한계비율 = 값 / 한계_2 \quad (1)$$

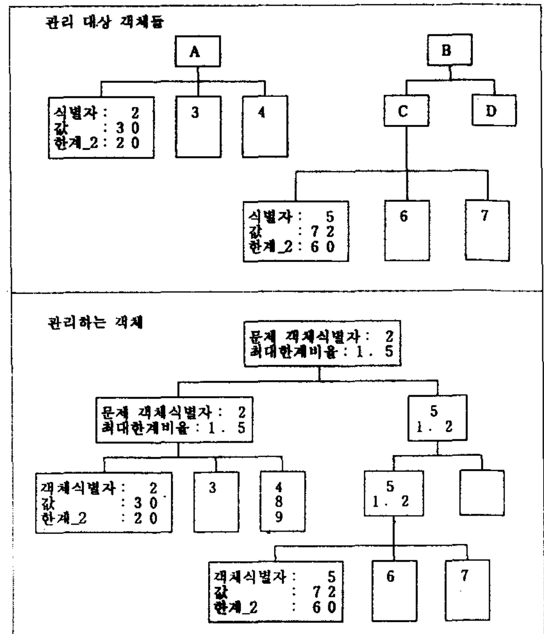
〈표 8〉 클래스의 성능 데이터

〈Table 8〉 Performance data of the class

이름	유형	설명
객체식별자	Integer	한계비율이 가장 큰 데이터의 객체 식별자
데이터식별자	Integer	한계비율이 가장 큰 데이터의 식별자
데이터값	Float	최대 한계비율
객체수	Integer	한계값을 초과한 객체의 수
데이터수	Integer	한계값을 초과한 데이터의 수

(그림 6)은 관리하는 객체가 관리 대상 객체들의 성능 데이터로부터 〈표 8〉의 클래스 성능 데이터를 설정하는 과정이다. 위쪽의 두개의 트리 구조는 자원 관리자 클래스와 실체화된 관리 대상 객체들, 즉 리프 노드들의 관계를 보여준다. 리프 노드들은 관리하는 객체로부터 부여받은 객체 식별자를 가지고 있다. 아래쪽은 관리하는 객체의 데이터베이스를 (그림 5)와 달리 표현한 것이다. 뿌리 노드로부터 하위 노드로 찾아가는 포인터와 (그림 5)에서와 같이 하위에서 상위로 찾아가는 포인터가 각 노드에 존재하나 그림에서 생략되었다.

예를 들어, 관리 대상 객체 "5"가 큐의 길이가 한계값을 넘어서 사건 발생 메시지를 관리하는 객체에게 보냈다고 가정하자. 관리하는 객체는 자신의 데이터베이스에서 "5"를 찾아서 값을 설정하고 한계비율을 계산하여 상위 클래스의 최대 한계비율과 비교하여 후자가 크면 멈추고, 전자가 크면 최대 한계비율을 수정한 다음, 차상위 클래스의 최대 한계비율과 다시 비교한다. 이와 같이 뿌리 노드까지 시행하고, 그 과정에서 〈표 8〉의 나머지 필드들도 설정된다. 또 관리 대상 객체 "2"의 활용도가 한계값을 넘어서 사건 발생 메시지를 관리하는 객체에게 보내면 "5"와 같은



(그림 6) 성능 데이터베이스에 클래스의 성능 데이터 적용

(Fig. 6) An example of the class performance database

방법으로 수행한다. (그림 6)에서는 뿌리 노드에서 "2"의 한계비율이 "5" 보다 크므로 뿌리 노드에는 "2"의 최대 한계비율이 설정된다. 이와 같은 구조로 관리하는 객체는 뿌리 노드나 중간 노드를 검색함으로써 하위 구조의 성능을 개괄적으로 파악할 수 있어서 불필요한 성능 데이터의 요구를 줄일 수 있다.

5. 성능 객체간의 프로토콜

관리 대상 객체와 관리하는 객체간에는 프로토콜 I, 관리하는 객체간에는 프로토콜 II가 있다. 프로토콜 I은 SNMP의 RFC 1157 및 ISO/CCITT의 ISO 9595/X.710인 CMIS(Common Management Information Service)[30]와 ISO 9596/X.711인 CMIP(Common Management Information Protocol)[31]과 유사하다. 그러나 상기 두 프로토콜은 네트워크 관리를 위한 것이기 때문에 시스템 자원 및 사용자와 성능 관리자간의 프로토콜을 정의하는 경우와는 상이한 점이 있다. ISO/CCITT ISO 9595/X.710은 사용자 인터페이스로 사용되는 일곱 가지 서비스를 정의하고 있는데 본 논문에서는 유사한 개념의 경우 ISO 용어를 그대로 사용하였다.

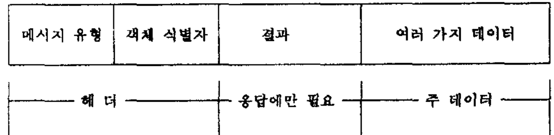
5.1 서비스의 정의

대화 형식은 서비스의 요청 후 응답이 있어야 다음 요청을 할 수 있는 대화식 프로토콜인 반이중 플립플롭(half-duplex flip-flop)이다. 〈표 9〉는 서비스의 종류를 보여주고 있고, MO는 관리하는 객체를, mo는 관리 대상 객체를 약칭한 것이다.

위의 서비스 중 분산 환경에 사용되는 MIGRATE와 INFORM을 제외한 서비스에 대한 메시지 형식을 다음절에서 논의하겠다.

5.2 프로토콜 데이터 형식

각 서비스는 요청과 응답의 형식을 갖는다. CREATE를 제외한 프로토콜 데이터는 일반적으로 다음과 같이 헤더(header)와 주 데이터로 나눌 수 있다. 헤더 데이터에는 메시지 유형과 객체 식별자가 들어가고 주 데이터에는 여러 가지 메시지 유형에 맞는 데이터가 들어간다.



〈표 9〉 서비스의 종류
〈Table 9〉 Services

서비스	프로토콜	설명
CREATE	I	mo가 자신의 클래스/객체/속성 등을 MO에 등록 요구
DELETE	I	MO/mo가 상대방에게 클래스/객체/속성의 등록 취소 요구
GET	I	MO가 mo에게 성능 데이터를 요구
SET	I	MO가 mo의 설정 가능한 속성의 수정을 요구
ACTION	I	MO가 mo에게 제어 명령을 보냄
CANCEL	I	MO가 mo에게 요청한 일의 수행 취소를 요구
EVENT-REPORT	I	mo가 MO에게 사건을 자발적으로 보고
GET-NEXT	I	MO가 mo에게 다음 성능 데이터를 요구
AUTO-REPLY	I	MO가 mo에게 주기적으로 성능 데이터를 요구
STATUS-REPORT	I	mo의 값이 변할 때마다 MO에게 자발적인 메시지 전송
MIGRATE	II	MO가 다른 MO에게 mo 데이터를 이전(MO의 부하 균형)
INFORM	II	MO간의 정보교환(SNMPv2의 inform-request와 유사)

프로토콜 I에 해당하는 메시지의 프로토콜 데이터 단위(Protocol Data Unit)를 ASN.1 (Abstract Syntax Notation One)[32] 형태로 나타낸 것을 부록에 수록하였다.

6. 구현 및 고려사항

성능 관리에 따른 부하[1, 33, 34]를 최소화하면서 정확한 정보를 수집하고, 시스템을 객체 지향으로 설계하고 구현함으로써 데이터의 무결성, 객체의 재사용 등 객체 지향이 주는 장점을 최대화해야 한다. 자원 관리자는 사용자의 요청을 처리하는 도중, 적절한 지점에서 자신의 성능 데이터를 수집하므로 모니터링 부하는 요청 처리에 비해 무시될 수 있고 객체 사이의 대화 메시지의 수를 줄이는 것이 중요하다. 관리 대상 객체(이하 mo라 함)와 관리하는 객체(이하 MO라 함)의 인터페이스가 정의되어 있고 MO는 mo가 M-CREATE시 등록하는 어떤 성능 데이터도 수용할 수 있으므로 MO를 독립적으로 개발할 수 있다.

본 논문에서는 자원 관리자가 관리 대상 객체로서 성능 관리에 능동적으로 참여함을 가정하였다. 일반적으로 자원 관리자는 운영체제나 서브시스템의 일부분이므로 이를 변경하는 것은 쉽지 않으나 3 장에서 정의한 대리자를 이용해서 구현해 볼 수 있다. 본 논문에서는 관리 대상 객체로서의 응용 프로그램이 계측을 통해 성능 관리에 참여함을 보이고자 한다.

6.1 사용자의 설계

응용 프로그램의 성능은 주로 계측을 통해 이루어져 왔다. 모듈, 프로시쥬어 또는 블록 등의 수행시간 및 시스템 함수 호출을 통해 시스템의 자원을 엑세스하는 시간과 DBMS, OLTP 등 서브시스템의 요청 처리시간도 모니터 대상이 된다. 구조적 프로그램의 경우 DO...WHILE, IF...THEN ELSE 등의 단위 수행시간도 모니터 대상이 될 수 있고 고급언어로 작성된 프로그램에 대해 문장 단위로 계측할 수도 있으나, 페이징이나 시분할 스케쥴링 등의 영향으로 응용 프로그램의 수행시간은 가변적이므로 온라인 모니터에는 부적절하다. 시스템 함수를 호출한 후 대기상태에 들어간 사용자는 실제로 자원을 사용하는 시간과 자원을 사용하기 위해 대기하는 시간을 구분할 수 없으며, 사용자는 자원의 사용을 위해 대기한 시간의 수행시간에 대한 비율을 최소화해야 한다.

6.1.1 정적 계측

사용자는 구현하기 쉽게 정적으로 계측하였으며 프리-컴파일러 및 프리-링커를 도입하였다. 프리-컴파일러는 원시 코드로부터 계측 장소를 찾아서 계측 코드를 삽입하여 컴파일러에 넘겨줄 수정된 원시 코드를 만들고 구체적인 계측 장소 정보, 즉 "계측 순번, 함수 이름 또는 라인 수, 해당 라인에 표시된 주석 등", 을 포함하는 계측 파일을 만든다. 이러한 함수 이름이나 주석은 성능 데이터의 이름으로 사용된다. 또 주 프로그램의 프리-컴파일시, 계측 파일을 읽어서 성능 메트릭을 보관할 기억장소를 얻고 관리 대상 객체를 생성하는 스텁 루틴(stub routine)이 프로그램의 첫머리에 추가된다. 계측 장소는 〈표 10〉과 같이 매개변수 파일에 기술되어 프리-컴파일러에 주어진다. 프리-링커는 링커의 매개변수에 명시된 모듈의 순서대로 계측 파일을 읽어서 그 내용에 모듈 이름을 삽입하여 새로운 계측 파일을 만든다. 일반적으로 정적인 계측은 계측점 주위에 계측 코드를 삽입한다. 본 논문에서는 여기에 스텁 루틴과 관리 대상 객체 역할을 하는 모듈을 추가하여 성능 관리자가 응용 프로그램의 성능을 대화하면서 측정할 수 있게 했다.

〈표 10〉 계측 장소
〈Table 10〉 Instrumentation locations

표시	내용
call	모든 함수 호출
syscall	시스템 함수 호출
usercall	사용자 함수 호출
do_while	do while 구조
do_until	do until 구조
라인 수	해당 라인의 call, do while 또는 do until

(그림 7)은 프로그램과 매개변수 파일을 입력으로 프리-컴파일러와 프리-링커를 수행하여 최종 계측 파일을 만드는 과정이다. 주 프로그램 A가 50번째 줄에서 B를 호출하고 65번째 줄과 80번째 줄에서 각각 시

스템 호출 malloc( )과 kill( )을 한다. 또 함수 B는 다시 함수 C를 12번째 줄에서 호출하고 30번째 줄에서 시스템 호출을 한다고 하자. A에서 수행하는 모든 호출과 B에서 수행하는 사용자 함수 호출을 모두 계측하고자 한다. 이 경우 B는 사용자 함수만 호출만 계측하려하므로 B의 시스템 호출은 무시된다.

모듈명	A	B
메개변수 파일	call	usercall

↓ 프리-컴파일

모듈명	A	B
계측 파일	50, B	12, C
	65, malloc	
	80, kill	

↓ 프리-링커

	순번	호출명	모듈명
최종 계측 파일	0	B	A
	1	malloc	A
	2	kill	A
	0	C	B

(그림 7) 계측 파일
(Fig. 7) Instrumentation file

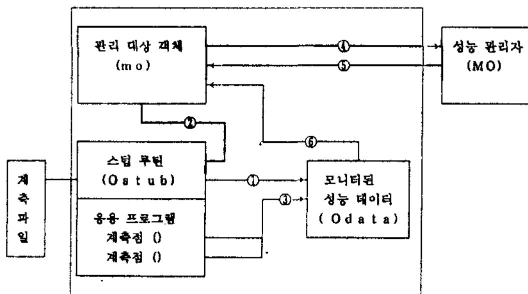
6.1.2 사용자의 성능 모니터

성능 관리를 위한 프리-컴파일러와 프리-링크를 거친 응용 프로그램은 (그림 8)과 같이 성능 관리 시스템에 참여한다. (그림 8)에서 스텁 루틴(Ostub)은 계측 파일을 읽어서 성능 데이터 장소(Odata)의 크기를 계산하여 msgget( )로 공유 기억장소를 얻고 초기화 한다①. 또 Ostub은 fork( )와 execve("mo", char *const argv[ ], char *const envp[ ])를 통해 관리 대상 객체(mo)를 작동시킨다②. 실행시 프로그램의 계측 코드들은 Odata의 해당 장소에 자신의 성능 데이터를 보관한다③. mo는 CREATE 메시지를 관리하는 객체인 성능 관리자에게 보내 자신의 성능 메트릭을 등록하고 성능 관리자는 (그림 5)와 같은 데이터베이스를 구축한다④. 성능 관리자가 성능 데이터를 GET.request 메시지를 통해 요구하면 mo는 Odata에 보관된 데이터⑥를 GET.response 메시지를 통해 전달한다⑤. 이때 ③과 ④, ⑤는 순차적으로 일어나지 않아도 된다. ④는 mo가 성능 관리자에게 요청하는 메시지의 통로를 의미하며, CREATE, DELETE 메시지가 여기에 속하고, 성능 관리자의 요청⑤에 대한 응답 메시지도 ④를 이용한다. ⑤는 성능 관리자가 mo에 요청하는 메시지의 통로로서, DELETE, ACTION, AUTO-REPLY, EVENT-REPORT, STATUS-REPORT, GET, SET, GET-NEXT, CANCEL 메시지 등이 여기에 해당되고 또한 사용자의 요청④에 대한 응답 메시지도 이용한다. Odata에서 해당 성능 데이터를 초기화①하고 보관③하며 추출⑥하는 과정은 계측점에서의 계측 코드를 설명함으로써 이해될 수 있다.

Ostub와 계측 루틴을 의사코드로 나타내면 다음과 같다.

```
Ostub( )
{
  Odata_pointer를 전역 변수로 선언;
  계측 파일을 사용해서 Odata의 크기를 계산한다.
  Odata를 공유 기억장소에 얻고 그 주소를 Odata_
  pointer에 넣는다.
  Odata를 초기화한다. 즉 함수 이름을 <표 1>의 "이
  름"에 넣고, "식별자"를 할당한다.
  모듈명을 Odata에 보관한다.
  fork( )한다.
  execve("mo", char *const argv[ ], char *const envp
  []) 하여 공유 기억장소의 키를 매개변 수로 관리 대
  상 객체 mo를 부른다.
}

호출전의 계측( )
{
  만약 slot_pointer가 유효하지 않으면,
    Odata_pointer로부터 순번과 모듈명을 이용해서
    Odata 내의 해당장소를 찾아서 넣는다.
    (이 장소는 <표 2>의 구조이다.
  현재의 시간을 얻어서 보관한다.
}

호출후의 계측( )
{
  현재의 시간을 얻는다.
```

이 시간에서 호출전의 보관된 시간을 빼서 〈표 1〉의 "값"에 보관한다.
"빈돗수"에 1을 더한다.
만약 "값"이 "최소값"보다 적거나 "최대값"보다 크면 최소 또는 최대값을 갱신한다.
"평균값"을 계산해 넣는다.
}



(그림 8) 실행중인 응용 프로그램의 모니터
(Fig. 8) Monitoring of a running program

6.2 프로토콜의 구현

객체간의 대화는 UNIX의 경우 메시지 큐를 이용하여 구현할 수 있다. 관리 대상 객체는 CREATE 요청의 응답으로 관리하는 객체로부터 유일한 객체 식별자를 부여받은 후, 이 객체 식별자를 통해 관리하는 객체와 대화한다. CREATE 시에는 관리 대상 객체는 응답을 받을 객체 식별자를 아직 부여받지 못하였으므로 자신의 메시지 큐를 만들고 키(key)를 메시지에 포함시켜서 관리하는 객체에게 보낸다. 관리하는 객체는 데이터베이스에 성능 프로토타입을 등록하고, 수신한 키를 이용하여 메시지 큐에 접속(msgget)한 후, 응답 메시지를 전송한다. 응답 메시지를 받은 관리 대상 객체는 CREATE을 위해 자신이 만든 메시지 큐를 제거한다. 이 응답 메시지에는 관리하는 객체가 관리 대상 객체에 부여한 시스템 내에 유일한 식별자가 있어서, 앞으로의 객체간의 대화에 이용된다. 객체간의 메시지 교환 프로토콜은 다음과 같이 의사코드로 표현될 수 있다.

managed_object( ) /* 관리 대상 객체 */
{
① 약속된 키를 사용하여 관리하는 객체의 메시지 큐를 접속한다(msgget 호출).
② 임의의 키를 사용하여 자신의 메시지 큐를 만든다(msgget 호출).
③ 이 메시지 큐의 키를 포함하는 CREATE. request 메시지를 완성하여, 관리하는 객체의 메시지 큐를 통해 메시지를 관리하는 객체에 보낸다(msgsnd 호출).
④ 응답을 자신의 메시지 큐에서 기다리며 수신한다(msgrcv 호출).
....
⑤ 자신의 메시지 큐를 제거한다(msgctl 호출).
(이후로는 관리하는 객체의 메시지 큐를 사용한다)
⑥ 응답에 포함된 식별자를 보관한다.
⑦ 자신이 성능 데이터를 수집하면서, 관리하는 객체의 메시지 큐에서 유형이 자신의 식별자인 메시지를 기다리며, 수신한다(msgrcv 호출)
....
⑧ GET, SET, AUTO-REPLY 등 정의된 메시지를 수신하여, 관리하는 객체의 메시지 큐를 통해 응답한다. EVENT-REPLY 등 자발적인 메시지도 송신한다.
⑨, ⑦, ⑧을 반복 수행한다.
}

Managing_Object( ) /* 관리하는 객체 */
{
① 약속된 키를 사용하여 메시지 큐를 만든다(msgget 호출).
② 자신의 큐에서 임의의 메시지를 기다리며 수신한다(msgrcv 호출)
....
③ 수신된 메시지 유형에 따라 수행한다.
1) CREATE 메시지이면,
[1] 메시지의 내용에 있는 송신자, 즉 관리 대상 객체의 키를 알아서 관리 대상 객체의 메시지 큐를 접속한다(msgget 호출).
[2] 관리 대상 객체의 성능 데이터 프로토타입을 데이터베이스에 만든다.
[3] 시스템 내에서 유일한 식별자를 만들어서 관리 대상 객체의 큐를 통해 전송한다

(msgsnd 호출).

4 ②를 수행한다.

2) EVENT-REPLY 메시지이면,

1 〈표 8〉의 클래스 성능 데이터에 한계값을 넘긴 객체의 개수 및 데이터의 개수를 증가시킨다. 한계값과의 비율이 보관된 최고값보다 크면 이 한계값을 최고값으로 하고 객체 및 데이터의 식별자를 저장한다.

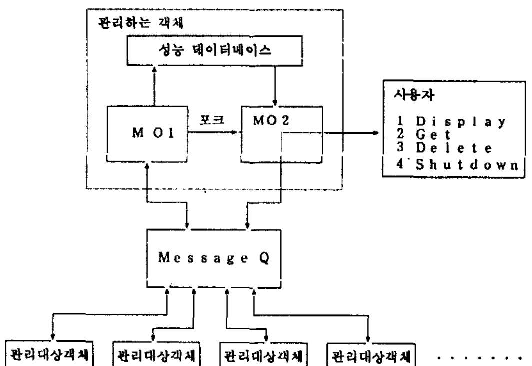
2 GET, AUTO-REPLY 등의 메시지를 이용해 충분한 성능 정보를 수집하거나 사용자에게 이 사실을 전한다.

3 ②를 수행한다.

3) ....

}

관리하는 객체는 관리 대상 객체로부터의 요청이나 응답을 대기해야하고 또 사용자 인터페이스로부터의 요청도 대기해야하므로 (그림 9)와 같이 두개의 프로세스로 구성되어야 한다. 관리 대상 객체는 (그림 8)과 같이 구성함으로써 관리하는 객체와의 대화와 실제 업무의 수행을 병행할 수 있다. MO1은 메시지 큐를 통해 받은 성능 데이터를 공유 기억장소에 저장하고, 사용자 인터페이스를 제공하는 MO2는 이 기억장소로부터 데이터를 읽어서 사용자에게 제공한다. 객체간에 CREATE, DELETE, GET 및 EVENT-REPORT의 요청과 응답 메시지를 지원하고, 성능 데



(그림 9) 메시지 큐를 이용한 객체간 대화 구성도
(Fig. 9) Conversation between the objects using message queue

이터베이스의 내용을 조회하거나 GET 또는 DELETE의 요청은 사용자 인터페이스를 통해 할 수 있게 하였다.

MO와 mo사이의 데이터 교환은 프로세서의 부하를 크게 유발할 수 있으므로 mo는 성능 데이터를 공유 기억장치의 정해진 장소에 저장하고 MO는 이 장소를 읽어서 성능 데이터를 얻는 방법을 고려해 볼 수 있다. mo는 M-CREATE 메시지에 데이터가 저장될 주소를 데이터의 다른 속성과 함께 포함시키고, MO는 이 주소로부터 성능 데이터를 읽는다. 이러한 모델[8]은 부하를 줄일 수 있을 뿐만 아니라 mo와 MO가 중복으로 가지는 데이터를 단일화할 수 있다. 그러나 데이터가 객체 외부에 있고 객체의 메소드를 통하지 않고 직접 접근이 가능하므로 객체 지향 모델은 아니다. mo와 MO 사이에는 MO가 데이터를 대기(wait)하고, mo는 데이터를 저장한 다음 MO를 포스트(post)하는 대기/포스트 관계를 가질 수도 있다.

6.3 프로토콜의 부하

자원 관리자는 수행 중에 자신의 성능 데이터를 수집하고 있다. 예를 들어 디스크 구동기는 큐의 길이나 액세스 횟수와 같은 성능 데이터를 가지고 있을 뿐만 아니라 활용도 등을 쉽게 추출할 수 있다. 그러므로 본 논문의 모델을 만족하는 자원 관리자가 성능 데이터를 수집하는 부하는 수행 과정에서 흡수되므로 무시해도 된다. 그러나 다수의 관리 대상 객체가 관리하는 객체와 성능 데이터를 교환해야 하므로 메시지 교환으로 인한 시스템의 부하는 최소화되어야 한다. EVENT-REPORT, STATUS-REPORT, GET-NEXT 및 AUTO-REPLY 등은 메시지 교환을 줄이기 위해 도입된 서비스들이다.

UNIX의 메시지 큐로 구현하는 경우, 메시지 교환으로 인한 부하는 〈표 11〉과 같이 얻을 수 있는데 실험에서는 평균 1,000 바이트의 메시지를 송수신하는데 걸리는 시간을 측정하였다. 큐의 대기자 수는 메시지를 전송하거나 수신을 요청할 때 이미 큐에 대기하고 있는 메시지의 수를 뜻한다. 측정 결과, 〈표 11〉에서 보는바와 같이 송수신 시간은 대기자의 수에 따라 대략 같은 비율로 증가한다.

〈표 11〉에서 Unisys의 경우 메시지의 송수신에 약 0.22 ms 정도 걸리므로 초당 약 4,545개의 메시지를

〈표 11〉 메시지 큐를 통한 메시지의 평균 송수신 시간
〈Table 11〉 Average message transmission time through message queue

하드웨어	운영체제	큐의 대기자 수			
		0	4	9	19
Unisys U/6000	SVR4.2	0.22 ms	1.4 ms	2.6 ms	4.9 ms
HP D350/2	HP-UX 10.10	0.06 ms	0.5 ms	0.7 ms	1.3 ms
Sun Server 1000	Solaris 2.4	0.06 ms	0.4 ms	0.7 ms	1.4 ms

송수신할 수 있다.

$$1,000 \div 0.22\ \text{ms} = 4,545\ \text{회/초} \quad (2)$$

일반적으로 상용 온라인 성능 모니터는 모니터로 인한 부하를 2-5% 정도 허용하고 있다. 여기서 객체가 성능 데이터를 수집하는 부하를 무시하고 메시지의 전송에 시스템의 프로세서 용량 중 1 %를 할당한다고 가정하면, 4,545 × 0.01 = 45 회/초, 즉 초당 45회 정도의 메시지를 송수신할 수 있다.

메시지 도착율이 지수분포를 이루고 서비스 시간을 상수로 볼 때 M/D/1 큐를 적용할 수 있고, 전송 밀도는 다음과 같다.

$$\text{전송 밀도}(\rho) = \text{도착율}(\lambda) \times \text{작업당 서비스 시간}(E[s]) = 45 \times 0.22 \times 0.001 = 0.01 \quad (3)$$

n 개의 사용자가 큐에 존재할 확률 Pn은 다음 세개의 식으로 나타낼 수 있다[1].

$$P_n = \begin{cases} 1-\rho & , n=0 \\ (1-\rho)(e^{\rho}-1) & , n=1 \\ (1-\rho)\sum_{j=0}^{n}\frac{(-1)^{(n-j)}(j\rho)^{(n-j-1)}(j\rho+n-j)e^{j\rho}}{(n-j)!} & , n\geq 2 \end{cases} \quad (4)$$

전송 밀도를 위 식에 대입하면, 큐에 사용자가 없을 확률은 0.99, 사용자가 하나일 때는 0.0099가 된다. 그러므로 사용자가 둘 이상일 확률(1 − 0.99 − 0.0099)은 0.00001 이다. 이것은 대단히 적은 값이므로 〈표 11〉에서 보는 바와 같이 대기자 수가 증가함으로 생기는 메시지 큐의 성능 저하는 무시할 수 있다.

7. 결 론

자원 관리자와 사용자를 관리 대상 객체로 정의하여 성능 관리에 능동적으로 참여케하고 관리하는 객체와의 대화 프로토콜을 정의함으로써 성능 관리 환경을 표준화할 수 있었다. 객체간의 대화 표준화와 더불어 성능 메트릭을 분류함으로써 성능 관리 시스템의 이식성과 상호 작용성을 높혔으며 객체를 독립적으로 개발할 수 있게 하였다. 또 정적인 계측을 통해 사용자 즉 응용 프로그램이 관리 대상 객체로 구현될 수 있음을 보였다. 사용자는 시스템 함수 호출을 통해 자원 관리자에 접근하므로 시스템의 각 함수가 논리적 자원으로 정의되면 사용자와 자원 간의 관계를 종합적으로 파악할 수 있다.

성능 관리를 위한 전문가 시스템을 만들기 위해서는 성능 데이터간의 상관 관계, 운영체제와 그 서브시스템 간의 관계, 논리적 자원과 물리적 자원의 관계, 객체간의 대화 프로토콜의 확장 등이 더 연구되어야 할 것이다. ISO/CCITT 표준을 성능 관리자간에 적용하면 분산 환경에서의 시스템 성능 관리를 쉽게 구현할 수 있을 것이다.

참 고 문 헌

[1] R. Jain, 'The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling', John Wiley & Sons. Inc., 1991.

[2] R. Snodgrass, "A relational Approach to Monitoring Complex Systems", ACM Trans. Computer Systems, Vol.6, pp.156-196, May 1988.

[3] Beth A. Schroeder, "On-Line Monitoring: A Tutorial", IEEE Computer, pp.72-78, Jun. 1995.

[4] W. Gu, J. Vetter, K. Schwan, "An Annotated Bibliography of Interactive Program Steering", SIGPlan Notices, Vol.29, No.9, pp.140-148, Sep. 1994.

[5] S. Mullender, ed., "Distributed Systems, 2nd ed.", ACM Press, pp.283-312, New York, 1993.

[6] M. Kaelbling, D. Ogle, "Minimizing Monitoring Costs: Choosing Between Tracing and Sampling",

Proc. 23rd Int'l Conf. System Sciences, IEEE CS Press, pp.314-320, Los Alamitos, CA, Jan. 1990.

[7] M. Loukides, 'System Performance Tuning', O'Reilly & Associates. Inc., 1991.

[8] 김용수, 이금석, "객체 지향 온라인 성능관리 시스템의 설계", 정보과학회논문지(A) 제23권,제6호, pp.594-608, 1996년 6월.

[9] IBM, "MVS/XA Resource Measurement Facility (RMF) reference and user's guide", IBM, 1985.

[10] Candle, "Omegamon II for MVS: Volume II Advanced Reference Realtime Component", Candle, 1991.

[11] Boole & Babbage, "MV Manager for MVS Reference Manual", Boole & Babbage, 1993.

[12] Landmark, "The Monitor for MVS: Version 1.3 Reference Manual", Landmark, 1994.

[13] Robert Berry and Mark Maccabee, "An Object-Oriented Data Model for the Automation of Computer Performance Management", CMG'91 Proceedings, pp.213-223, 1991.

[14] Robert Berry and Seetha Lakshmi, "SystemView Design Considerations for a Real-time Performance Management Application", CMG'92 Proceedings, pp.186-192, 1992.

[15] Yong-Soo Kim, Keum-Suk Lee, "Object Oriented Online Performance Management System", 22nd Euromicro Conference, IEEE CS Press, pp.95-100, Prague, Czech, Sep. 1996.

[16] 김용수, 이금석, "사용자 주소공간에서 DBMS 응답시간의 온라인 모니터", 정보과학회 '93 가을 학술발표논문집, 제20권, 2호, pp.218-221, 1993.

[17] T. Ball, J. R. Larus, "Optimally Profiling and Tracing Programs", 19th ACM Symposium on Principles of Programming Languages, Albuquerque, NM, Jan. 19-22, 1992.

[18] MIPS, "RISCompiler Languages Programmer's Guide", MIPS Computer Systems, Inc., Dec. 1988.

[19] Rolf Borgeest, Bernward Dimke, Olav Hansen, "A trace based performance evaluation tool for parallel real time systems", Parallel Computing, Vol.21, No.4, pp.551-564, Apr. 1995.

[20] Barton P. Miller et al., "The Paradyn Parallel Performance Measurement Tools", IEEE Computer 28, Nov. 1995.

[21] Jeffrey K. Hollingsworth, Barton P. Miller, "Dynamic Control of Performance Monitoring on Large Scale Parallel Systems", 7th ACM International Conference on Supercomputing, Tokyo, Jul. 1993.

[22] J. K. Hollingsworth, R. B. Irvin, B. P. Miller, "The Integration of Application and System Based Metrics in A Parallel Program Performance Tool", 1991 ACM SIGPLAN Notices Symposium on Principles and Practice of Parallel Programming, Apr. 1991.

[23] J. K. Hollingsworth, B. P. Miller, J. Cargille, "Dynamic Program Instrumentation for Scalable Performance Tools", 1994 Scalable High Performance Computing Conference, Knoxville, Tenn., 1994.

[24] K. Schwaan, R. Ramnath, S. Vasudevan, D. M. Ogle, "A Language and System for Parallel Programming", IEEE Transactions on Software Engineering, Apr. 1988.

[25] P. C. Bates, J. C. Wileden, "DEL: A Basis for Distributed System Debugging Tools", 15th Hawaii International Conference on System Sciences, Jan. 1982.

[26] B. Brugge, "A Portable Platform for Distributed Event on Parallel and Distributed Debugging", SIGPLAN Notices, Dec. 1991.

[27] W. Gu et al., "Facon: Online Monitoring and Steering of Large-Scale Parallel Programs", Tech. Report GIT-CC-94-21, College of Computing, Georgia Institute of Technology, Atlanta, Ga., 1994.

[28] Werner Dirlewanger, "Measurement and rating of performance of computer-based software systems", ISO/IEC JTC1/SC7 Evaluations and Metrics, May 1996.

[29] ISO(CCITT), ISO/IEC 10164-11(X. 739): Workload

Monitoring Function

[30] ISO(CCITT), ISO/IEC 9595(X.710): Common Management Information-Service Definition, 1991.

[31] ISO(CCITT), ISO/IEC 9596(X. 711): Common Management Information-Protocol Definition, 1991.

[32] ISO(CCITT), ISO/IEC 8824(X.208): Abstract Syntax Notation One, 1991.

[33] Clement H. C. Leung, 'Quantitative Analysis of Computer Systems', John Wiley & Sons, 1988.

[34] K. M. Chandy, C. H. Sauer, "Approximate Solution of Queuing Models", IEEE Computer, 13, No.4, pp.25-32, 1980.

부 록

```
Performance_PDUs ::= BEGIN
IMPORTS TimeTicks FROM RFC1155-SMI;
PDUs ::= CHOICE { create_request [0] IMPLICIT CreateRequest,
                  create_response [1] IMPLICIT CreateResponse,
                  delete_request [2] IMPLICIT DeleteRequest,
                  delete_response [3] IMPLICIT DeleteResponse,
                  get_request [4] IMPLICIT GetRequest,
                  get_response [5] IMPLICIT GetResponse,
                  set_request [6] IMPLICIT SetRequest,
                  set_response [7] IMPLICIT SetResponse,
                  get_next_request [8] IMPLICIT GetNextRequest,
                  get_next_response [9] IMPLICIT GetNextResponse,
                  auto_reply_request [10] IMPLICIT AutoReplyRequest,
                  auto_reply_response [11] IMPLICIT AutoReplyResponse,
                  action_request [12] IMPLICIT ActionRequest,
                  action_response [13] IMPLICIT ActionResponse,
                  cancel_request [14] IMPLICIT CancelRequest,
                  cancel_response [15] IMPLICIT CancelResponse,
              event_report_request [16] IMPLICIT EventReportRequest,
              event_report_response [17] IMPLICIT EventReportResponse,
              status_report_request [18] IMPLICIT StatusReportRequest,
              status_report_response [19] IMPLICIT StatusReportResponse }

CreateRequest ::= SEQUENCE { createRequestMessage MessageType(1),
                             version    VersionNumber,
                             key        MessageQueueKey,
                             classCount  CountType,
                             classIdentifiers OBJECT IDENTIFIER,
                             performanceData PerformanceData }
MessageType ::= INTEGER(0 .. 255)
VersionNumber ::= INTEGER(0 .. 32767)
```

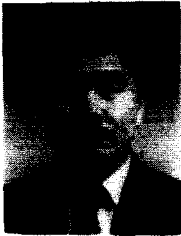
```
MessageQueueKey ::= INTEGER
CountType ::= INTEGER(0 .. 32767)
PerformanceData ::= SEQUENCE { dataCount CountType,
                               dataProtoType SEQUENCE OF DataProtoType }
DataProtoType ::= SEQUENCE { dataIdentifier IdentifierType,
                             dataname Name,
                             type DataType,
                             limitValue Value,
                             nomalValue Value,
                             eventStatus EventStatus }
IdentifierType ::= INTEGER
Name ::= VisibleString
DataType ::= INTEGER(0 .. 255) { type_1(1),
                                 type_2(2),
                                 type_3(3) }
Value ::= CHOICE { integer INTEGER,
                   real   REAL }
EventStatus ::= INTEGER(0 .. 255) { eventReport(1),
                                    noEventReport(2),
                                    statusReport(3),
                                    noStatusReport(4) }

CreateResponse ::= SEQUENCE { createResponseMessage MessageType(2),
                              version   VersionNumber,
                              objectIdentifer IdentifierType,
                              createResult  CreateResult }
CreateResult ::= INTEGER(0 .. 255) { noError(0),
                                     versionError(1),
                                     formatError(2),
                                     overHeadError(3),
                                     spaceError(4) }

DeleteRequest ::= SEQUENCE { deleteRequestMessage MessageType(3),
                             objectIdentifier IdentifierType,
                             downReason  DownReason }
DownReason ::= INTEGER(0 .. 255) { deleteManagedObject(1),
                                   deleteManagingObject(2),
                                   overHeadMessage(3),
                                   overHeadCollection(4),
                                   protocolError(5) }

DeleteResponse ::= SEQUENCE { deleteResponseMessage MessageType(4),
```

```
                                    objectIdentifier IdentifierType,
                                    deleteResult DeleteResult }
DeleteResult ::= INTEGER(0 .. 255) { noError(0),
                                     requestFailed(5) }

GetRequest ::= SEQUENCE { getRequestMessage MessageType(17),
                          objectIdentifier IdentifierType,
                          requestStructure RequestStructure }
RequestStructure ::= SEQUENCE { dataCount CountType,
          requestDataStructure SEQUENCE OF RequestDataStructure }
RequestDataStructure ::= SEQUENCE { dataIdentifier IdentifierType,
                                    attributeData AttributeData }
AttributeData ::= SEQUENCE { attributeCount CountType,
                     attributeNames SEQUENCE OF AttributeName }
AttributeName ::= VisibleString

GetResponse ::= SEQUENCE { getResponseMessage MessageType(18),
                           objectIdentifier IdentifierType,
                           getResult GetResult,
                           responseStructure ResponseStructure }
GetResult ::= INTEGER(0 .. 255) { noError(0),
                                  noSuchIdentifier(6),
                                  noSuchName(7),
                                  badValue(8),
                                  genError(9) }
ResponseStructure ::= SEQUENCE { dataCount CountType,
        responseDataStructure SEQUENCE OF ResponseDataStructure }
ResponseDataStructure ::= SEQUENCE { dataIdentifier IdentifierType,
                                     attributeStructure AttributeStructure }
AttributeStructure ::= SEQUENCE { attributeCount CountType,
                        attributePairs SEQUENCE OF AttributePair }
AttributePair ::= SEQUENCE { attributeName AttributeName,
                             attributeValue Value }

SetRequest ::= SEQUENCE { setRequestMessage MessageType(19),
                          objectIdentifier IdentifierType,
                          responseStructuree ResponseStructure }

SetResponse ::= SEQUENCE { setResponseMessage MessageType(20),
                           objectIdentifier IdentifierType,
                           setResult  SetResult,
                           dataIdentifier IdentifierType,
                           attributeOffset INTEGER }
```

```
SetResult ::= INTEGER(0 .. 255) { noError(0),
                                  noSuchIdentifier(6),
                                  noSuchName(7),
                                  badValue(8),
                                  genError(9),
                                  readOnly(10) }

GetNextRequest ::= SEQUENCE
                    { getNextRequestMessage MessageType(21),
                      objectIdentifier IdentifierType,
                      dataIdentifier IdentifierType }

GetNextResponse ::= SEQUENCE
                    { getNextResponseMessage MessageType(22),
                      objectIdentifier IdentifierType,
                      getNextResult  GetNextResult,
                      dataIdentifier IdentifierType,
                      attributePair  AttributePair }
GetNextResult ::= INTEGER(0 .. 255) { noError(0),
                                      noSuchIdentifier(6),
                                      genError(9) }

AutoReplyRequest ::= SEQUENCE
                    { autoReplyRequestMessage MessageType(23),
                      objectIdentifier IdentifierType,
                      replyCount CountType,
                      replyCycle TimeTick,
                      autoReplyRequestStructure RequestStructure }

AutoReplyResponse ::= SEQUENCE
                    { autoReplyResponseMessage MessageType(24),
                      objectIdentifier IdentifierType,
                      autoReplyResult GetResult,
                      replyCount CountType,
                      replyCycle TimeTick,
                      autoReplyResponseStructure ResponseStructure }

ActionRequest ::= SEQUENCE { actionRequestMessage MessageType(33),
                             objectIdentifier IdentifierType,
                             commandIdentifier IdentifierType,
                             actionParameter ActionParameter }
ActionParameter ::= SEQUENCE OF VisibleString
```

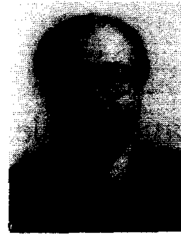
```
ActionResponse ::= SEQUENCE
                   { actionResponseMessage MessageType(34),
                     objectIdentifier IdentifierType,
                     actionResult ActionResult }
ActionResult ::= INTEGER(0 .. 255) { noError(0),
                                     noSuchCommand(11),
                                     parameterError(12) }

CancelRequest ::= SEQUENCE { cancelRequestMessage MessageType(35),
                             objectIdentifier IdentifierType,
                             messageType MessageType }

CancelResponse ::= SEQUENCE
                   { cancelResponseMessage MessageType(36),
                     objectIdentifier IdentifierType,
                     cancelResult CancelResult,
                     messageType MessageType }
CancelResult ::= INTEGER(0 .. 255) { noError(0),
                                     cancelFailed(13),
                                     alreadyDone(14) }

EventReportRequest ::= SEQUENCE
                       { eventReportRequestMessage MessageType(49),
                         objectIdentifier IdentifierType,
                         dataCount DataCount,
                         eventData SEQUENCE OF EventData }
EventData ::= SEQUENCE { dataIdentifier IdentifierType,
                         eventFlag EventFlag }
EventFlag ::= INTEGER(0 .. 255) { eventReport(1),
                                  noEventReport(2) }

EventReportResponse ::= SEQUENCE
                   { eventReportResponseMessage MessageType(50),
                     objectIdentifier IdentifierType,
                     eventReportResponseStructure ResponseStructure }

StatusReportRequest ::= SEQUENCE
                   { statusReportRequestMessage MessageType(51),
                     objectIdentifier IdentifierType,
                     dataCount DataCount,
                     statusData SEQUENCE OF StatusData }
StatusData ::= SEQUENCE { dataIdentifier IdentifierType;
```

```
                    statusFlag StatusFlag }
StatusFlag ::= INTEGER(0 .. 255) { statusReport(3),
                                   noStatusReport(4) }

StatusReportResponse ::= SEQUENCE
                 { statusReportResponseMessage MessageType(52),
                   objectIdentifier IdentifierType,
                   statusReportResponseStructure ResponseStructure }
END
```



김 용 수

1979년 부산대학교 문리대 물리학과 졸업(학사)
1978년~1987년 대한항공 시스템부 시스템 프로그래머
1986년 연세대학교 산업대학원 전자계산학(공학석사)
1997년 동국대학교 대학원 컴퓨터공학과(공학박사)
1997년~현재 현대정보기술주식회사 IS&TC 근무
관심분야: 운영체제, 시스템 성능관리



이 금 석

1971년 서울대학교 공과대학 응용수학과 졸업(학사)
1973년 한국과학기술연구소 전산개발센터 전산기술과 근무
1978년 한국과학기술원 전산학과 졸업(이학석사)
1981년~현재 동국대학교 컴퓨터공학과 교수
관심분야: 운영체제론, 컴퓨터 성능평가, 소프트웨어공학 등