소프트웨어 관리를 위한 확장 SNMP 에이전트

The Extended SNMP Agent for Managing Software

고은진*, 이길행*, 조국현**
ETRI*, 광운대학교 컴퓨터과학과**
{ejko,ghlee}@etri.re.kr, khcho@cs.kwangwoon.ac.kr

요 약

기존 망 관리에서 각 에이전트는 자신의 시스템에 있는 관리 대상 하드웨어 자원중심의 관리 정보를 정기적으로 수집하여 관리자에게 전달한다. 관리자는 전달된 관리 정보를 모아 분석하고 제어하는 방식으로 동작한다. 통신망의 기능이 다양해지고 복잡하게 발전하면서 각 노드에서 수행중인 소프트웨어의 비중이 커짐에 따라 망 구성 요소 중 하나인 소프트웨어를 고려하지 않은 망 관리는 정확한 망 상태를 제공하고 이에 따른 효율적인 조치를 취할 수 있는 기능을 제공하기가 어렵다. 또한 에이전트의 수가 증가함에 따라 모든 관리를 관리자가 수행해야 하는 기존 SNMP(Simple Network Management Protocol) 방식은 망 관리를 위한 데이터 트래픽이 증가함에 따른 망 전체 성능에 영향을 미치고 있다. 따라서 본 논문에서는 기존 망 관리 기능을 유지하면서 관리자의 위임을 받아 self-polling 기능과 에러 검출 기능을 수행하고 본 논문에서 정의한 소프트웨어 MIB을 통해 소프트웨어 자원을 망 관리 차원에서 제공하는 확장 SNMP 에이전트 모델을 제시한다.

1. 서 론

최근 통신망에 대한 사용자 요구사항의 다양성, 이를 수용하기 위한 통신망 기술의 복잡성 및 이에 따른 제반 기술의 발전 등 급격한 변화를 가져오고 있다. 이러한 통신망의 발전에 대처하는 대응 전략으로 요구되는 것이 통신망 관리 분야이다[5][12].

망 관리의 궁극적 목표는 망이 보다 효율적으로 동작할 수 있게 하면서 되도록 문제 발생 원인을 줄이는 일련의 작업들이다.

효율적인 관리를 위해 망 관리는 구성 관리, 성능 관리, 장애 관리, 보안 관리, 계정 관리등 다섯 가지 기능 영역으로 나눠 수행되고 있다[10].

이런 기능들과 더불어 망 관리에서 중요한 요소는 사용되는 망 관리 프로토콜이다. 망 관리 프로토콜은 인터넷 기반의 SNMP와 OSI 기반의 CMIP(Common Management Information Protocol)으로

나눌 수 있다[13]. CMIP은 SNMP에 비해 보다 강력하고 많은 기능을 제공하고 있으며 관리 대상 객체를 관리자가 생성, 삭제 할 수 있게 해주는 반면, SNMP는 CMIP보다 간단하기 때문에 다양한 기능을 제공하지 못하지만 구현하기 쉽고 이미 널리 사용되고 있다는 장점을 가지고 있다.

현재 통신망은 다양한 특성을 가지는 형태로 발전해감에 따라 각 노드에서 하드웨어뿐만 아니라 소프트웨어가 차지하는 비중이 증가하고 있다. 이에 따라 기존 하드웨어 중심의 관리에서 각 노드에서 동작 중인 주요 응용 프로그램까지 포함하여 일관된 형태로 관리할 필요성이 제기되고 있다[7]. 또한, 기존 망 관리에서 에이전트는 관리자의 요청에 의해 동작하는 수동적인 역할을 수행하고, 관리자는 모든 동작을 수행한 후 수집된 정보를 분석하는 역할까지 수행함으로써, 에이전트 수가 많은 경우에 관리자의 부담이 커질 수밖에 없어 효율적인 망 관리를 할 수 없다[1].

본 논문에서는 소프트웨어 관리와 관리자의 부담 감소라는 문제를 해결하기 위해 기존의 망 관리에 소프트웨어 자원 관리 기능을 추가하고 관리자 기능 중 에러 검색 기능과 폴링 기능을 대신 수행하는 에이전트 모델을 설계하고 구현하였다. 적용 대상 소프트웨어는 일반 소프트웨어보다 망에서 많이 사용되면서 망에 영향을 주는 서버급으로 한정하였다. 소프트웨어 관리 에이전트는 일반 에이전트의 기능에 소프트웨어 관련 기능을 추가하는 방법으로 구현하였으며 소프트웨어 MIB는 공통 부분과 개별 부분으로 나눠 정의하였다.

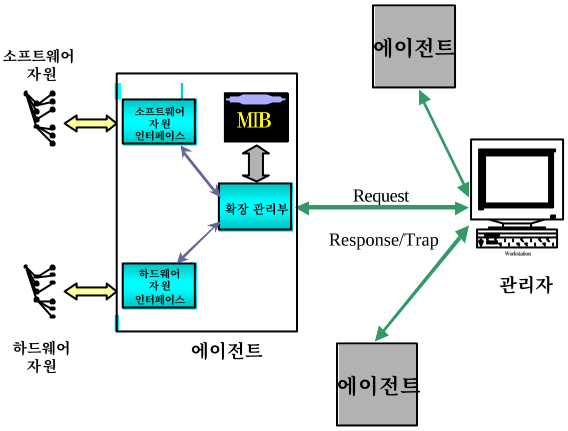
본 논문의 구성을 보면 2장에서 망 관리에 대해 살펴보고 3장에서는 소프트웨어 관리 기능을 가지며, 관리자 기능 일부도 가지는 소프트웨어 관리 에이전트 모델을 설계하고 4장에서는 3장에서 설계된 소프트웨어 관리 에이전트 모델을 구현한 결과를 보여주고 5장에서 결론을 맺는다.

2. SNMP 망관리

SNMP 망 관리 시스템은 관리 정보들의 모임인 MIB-II, 해당 관리 정보를 수집하여 관리 시스템의 질문에 응답하는 에이전트들과 이들로부터 받은 정보를 바탕으로 관리 기능을 수행하는 관리 스테이션, 그리고 이들 관리 시스템과 에이전트간 관리 프로토콜로 구성된다. SNMP와 MIB-II를 살펴보면 하드웨어 관리 중심임을 알 수 있다[9][11]. 하지만 많은 프로그램들이 클라이언트-서버 형태로 망에서 동작하고 있으며 망에서 소프트웨어의 비중이 증대하고 있다[7][8]. 즉 소프트웨어 관리를 고려하지 않은 경우에 문제가 발생하면 발생원인의 범위를 하드웨어 요소로 한정하여 해당 시스템의 소프트웨어에 문제가 있는 경우에 대한 처리를 할 수 없어 균형적인 망 관리를 하지 못했다[1]. 따라서 본 논문에서는 3장에서 소프트웨어 관리를 위한 MIB와 소프트웨어 에이전트 모델을 제안함으로써 기존 망 관리 시스템의 문제를 해결하고자 한다.

3. 소프트웨어 관리를 위한 확장 SNMP 에이전트

최근 들어 하드웨어 관리뿐만 아니라 소프트웨어 관리를 통한 균형적인 망 관리를 위한 여러 가지 연구가 진행되고 있다. 즉 통신망 환경에서 하위 계층 관리 일변도에서 벗어나 상위 계층 관리까지 망 관리 영역을 넓히고 있으며, 각 계층에 대한 일관된 망 관리가 필요하다. 또한 에이전트는 기본적으로 한 개의 하드웨어와 여러 개의 소프트웨어로 구성되며 각 소프트웨어들은 각각 다양한 특징들을 가지고 있고 한 관리자가 여러 에이전트들을 관리하기 때문에 각 에이전트에 있는 모든 관리 정보를 일괄적으로 관리하기에는 많은 오버헤드가 있게 된다[1]. 따라서 이번 장에서 기존 관리자의 오버헤드를 줄이기 위해 관리자의 기능 일부를 포함하면서 각종 서버 프로그램 관리 기능을 포함하여 일관된 망 관리 기능을 지원할 수 있는 에이전트를 설계하고자 한다.



(그림 1) 소프트웨어 관리를 위한 확장 SNMP 시스템 구성도

이를 위해 (그림 1)처럼 소프트웨어 관리 에이전트는 에이전트 MIB, 소프트웨어 자원 관리부, 하드웨어 자원 관리부와 확장 관리부로 구성된다. 확장 관리부의 요청이 있을 경우 소프트웨어 자원 인터페이스와 하드웨어 자원 인터페이스는 각 관리 대상 자원의 정보를 추출하여 이를 전달해 주는 기능을 수행하며, 확장 관리부는 주어진 정보를 토대로 소프트웨어 관리를 포함하는 관리 기능을 수행한다. 각 자원 관리부로부터 추출된 정보들은 MIB 형태로 에이전트 안에 저장된다.

3.1 소프트웨어 관리 MIB

소프트웨어는 종류가 다양하고 특징이 다르기 때문에 소프트웨어 MIB를 정의하고 이를 표준화 시키는 연구가 큰 진전을 보지 못하고 있다. 현재 개별 특성에 맞는 관리 객체들은 소프트웨어 개발 측에서 정의하는 방향과 일반적인 소프트웨어 정보에 관한 객체들을 정의하는 방향으로 연구가 진행 중이다[2][3][7].

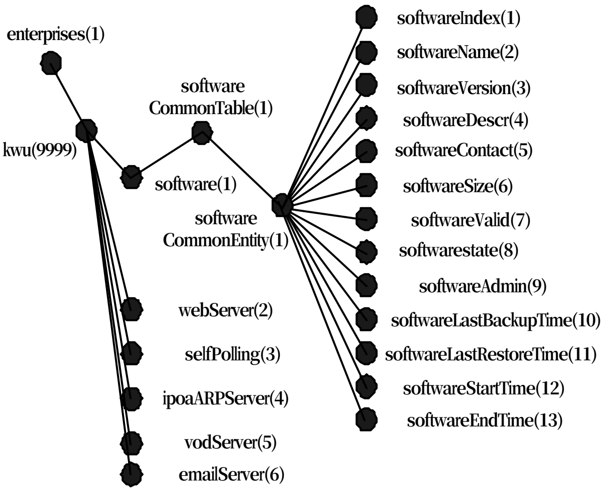
하지만 일관된 망 관리를 위해서는 각 소프트웨어의 공통되는 부분과 각 특성들을 함께 관리 객체로 정의하고 정의된 객체의 정보를 받아 관리해야 한다. 이를 위해 다음과 같은 MIB 설계 규칙에 따라 관리 객체를 정의할 필요가 생기게 된다.

MIB 설계는 우선 설계 대상의 특성을 파악하여 특성별로 관리 객체를 선언하고 값 특성에 따라 테이블을 만든다.

MIB는 표준 MIB와 기타 MIB가 있다. 개별적인 MIB는 모두 enterprises 아래에 새로운 객체 식별자(OID : Object Identifier)를 할당 받거나 등록되지 않은 객체 식별자 중에서 하나를 임의로 사용한다. 앞서 정의된 소프트웨어 관리 객체들에 식별자를 할당하여 enterprises 밑에 MIB 트리 구조를 만든다. 관리 객체의 유형, 접근 권한, 설명 및 객체 식별자는 ASN.1(Abstract Syntax Notation.1)을 이용한 SMI(Structure of Management Information) 규칙에 따라 MIB를 정의하게 되어 있다[4][6].

본 논문에서는 이러한 규칙에 따라 enterprises 아래에 kwu(9999)라는 임의의 객체 식별자를 정의하였으며 kwu(9999) 아래에 있는 객체들은 확장을 고려하여 SMIv2 규칙에 따라 정의되었다. 본 논문에서는 정의하고 있는 MIB들은 소프트웨어의 일반적인 사항에 대한 MIB와 특정 소프트웨어에 대한 MIB로 구분하여 따로 정의하는 것이 아니라, (그림 2)와 같이 두 MIB를 동시에 정의하여 사용하는 방법을 제안하고 있다.

즉 소프트웨어의 공통적인 사항과 관련된 관리 객체들을 software 밑에 softwareCommonTable이라고 하는 Table에 정의하고 특정 소프트웨어 MIB를 software 이후에 Web server, IPOAAARP server, VOD server, Email server를 각각 정의하고 있다.


enterprises(1)
kwu(9999)
software CommonTable(1)
software(1)
software CommonEntity(1)
webServer(2)
selfPolling(3)
ipoaARPServer(4)
vodServer(5)
emailServer(6)
softwareIndex(1)
softwareName(2)
softwareVersion(3)
softwareDescr(4)
softwareContact(5)
softwareSize(6)
softwareValid(7)
softwarestate(8)
softwareAdmin(9)
softwareLastBackupTime(10)
softwareLastRestoreTime(11)
softwareStartTime(12)
softwareEndTime(13)

(그림 2) 소프트웨어 MIB(1)

본 논문에서 각 소프트웨어의 공통사항으로 정의하고 있는 software-CommonTable을 살펴보면 softwareName은 소프트웨어의 이름을, software-Version은 해당 소프트웨어의 버전을 의미한다. softwareDescr는 소프트웨어에 대한 일반적인 사항을, software-Contact은 소프트웨어에 관한 사항을 문의하기 위한 인터넷 주소를 의미한다. softwareSize는 소프트웨어의 기본 크기를 나타낸다.

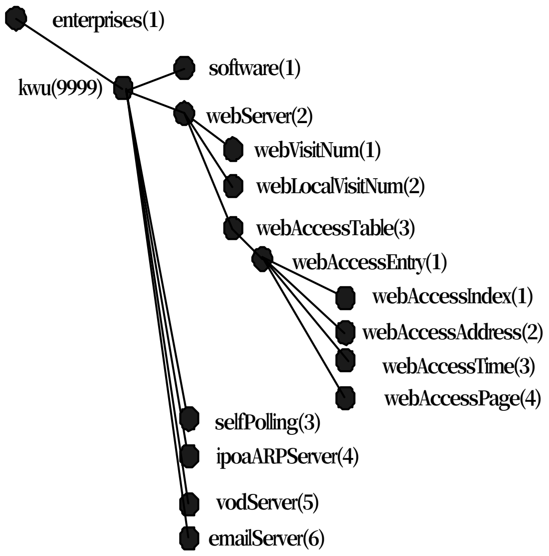
softwareValid는 소프트웨어가 유효기간이 지났는지를 의미하는데 ‘1’이 반환되면 유효함을 ‘2’가 반환되면 유효하지 않음을 나타낸다. Software-State는 현재 소프트웨어가 동작 중인지를 나타내며, softwareAdmin은 해당 소프트웨어 관리 책임자를 의미한다. SoftwareLastBackupTime은 가장 최근에 백업 시간을 의미하며, softwareLast-RestoreTime은 마지막으로 소프트웨어

를 복구한 시간을 의미한다.

softwareStartTime은 소프트웨어가 현재 동작 중일 경우 동작 시작 시간을, softwareEndTime은 가장 마지막 동작 후 종료된 시간을 의미한다. 이와는 별도로 각 소프트웨어마다 가지고 있는 특징에 대한 MIB는 softwareCommonTable과는 따로 정의되어야 한다. 따라서 본 논문에서는 Web Server, VOD Server, ipoaARPServer, emailServer에 대한 MIB를 정의하였고, 그 중에서 Web Server에 관한 MIB를 (그림 3)에서 보여주고 있다.

webserver에 관한 망 관리 객체들을 살펴보면 처음 두 개의 관리 객체는 scalar object type을 갖는다. WebVisitNum은 일정 기간 동안에 해당 웹 서버에 접속한 횟수를 말하며, webLocalVisitNum은 동일 기간동안에 로컬 호스트에서 접속한 횟수를 말한다. 본 논문에서의 일정 기간이란 15일을 말하며, 로컬 호스트는 IP 주소 네 자리 중 처음부터 세 번째 자리까지 동일한 호스트를 로컬로 정의하였다. 이 때 접속한 사용자의 주소가 같은 경우에는 한 번만 계산하였다. 나머지 3개의 관리 객체들은 vector object type을 가지며 webAccessAddress는 접속한 사용자의 주소를 의미하며 webAccessTime은 가장 최근 접속한 시간을 나타낸다. WebAccessPage는 가장 최근에 접속한 웹사이트 페이지를 의미한다.

selfPolling은 에이전트 자체 폴링을 위한 MIB다. 관리자가 폴링 변수를 설정해 주는데 Set 요청에서 해당 폴링 시간의 OID를 사용한다. Self-Polling 밑에는 다섯 개의 OID가 있으며 각각 3초, 5초, 1분, 5분 그리고 10분의 폴링 시간을 의미한다.


enterprises(1)
kwu(9999)
software(1)
webServer(2)
webVisitNum(1)
webLocalVisitNum(2)
webAccessTable(3)
webAccessEntry(1)
webAccessIndex(1)
webAccessAddress(2)
webAccessTime(3)
webAccessPage(4)
selfPolling(3)
ipoaARPServer(4)
vodServer(5)
emailServer(6)

(그림 3) 소프트웨어 MIB(2)

3.2 소프트웨어 자원 인터페이스

소프트웨어 자원 인터페이스는 확장 관리부로부터 관리 정보를 요구 받으면 해당 관리 정보를 가지고 있는 자원으로부터 정보를 추출하여 확장 관리부로 넘기는 기능을 수행한다. 이를 위해 MIB에 따라 해당 정보를 추출할 수 있는 함수를 이용한다. 이 때 해당 자원이 동작을 하지 않는 경우, 에러 메시지를 반환함으로써 이를 확장 관리부에 알리게 된다.

3.3 확장 관리부

확장 관리부는 기존 관리자의 기능 중 포트를 열고 PDU encode/decode 기

능과 하드웨어 정보 검색 기능 등의 일부 기능을 수행하면서 소프트웨어 관리 기능을 수행하기 위해 Transport Stack과 제어부, 확장 자원 관리부(XMP : eXtended Management Part), 폴링을 담당하는 SRPP(Self Retrieve Polling Part)와 PMIT(Polled-MO Information Table), 에러 처리를 담당하는 Fact Repository와 Event Processing Engine으로 구성된다. 각 부분의 자세한 기능은 다음과 같다.

3.3.1 Transport Stack과 제어부

Transport Stack은 port 161번을 열어 관리자와의 연결을 설정하는 부분이다. 관리자로부터 request가 PDU 형태로 오면 제어부에 넘기고 제어부에서 오는 response를 관리자에게 넘기는 기능을 수행한다[12].

제어부는 크게 세 가지 기능을 수행한다. 첫번째는 PDU encoding/decoding 기능이다. 두번째는 PDU에 대한 에러 검색 및 처리 기능이다. PDU를 decoding하는 동안에 전송 과정에서 발생한 에러를 발견하게 되면 해당 에러 처리 루틴을 사용하여 이를 처리하여 관리자에게 알려준다. 세번째는 관리자로부터 오는 request를 확장 자원 관리부나 SRPP(Self Retrieve Polling Part)에 전달하는 기능이다. 관리자로부터 오는 request는 크게 두 가지로 나눠진다. 즉, 하나는 일반적인 request이고 다른 하나는 SRPP request다. SRPP request는 자체 폴링을 수행하기 위해 PMIT의 변수를 설정하거나 변경하는데 사용된다. 따라서 제어부에서 OID를 가지고 request의 유형을 구분하여 처리하며, 이 때 request는 OID, value 혹은 OID 형태를 가진다. OID 형태는 Get/GetNext request와 관련된 것이고 OID, value 형태는 Set/SRPP request와 관련된 것이다. response는 확장 자원 관리부에서 OID, value 형태로 제어부로 온 후 PDU로 변경되어 진다. GetNext request와 같은 연속적인 요청을 처리하기 위해 제어부에서는 각 요청마다 sdlist에 관련 사항을 저장하고, 이후에 PDU를 만들어서 관리자에게 전송할 수 있게 해준다.

3.3.2 확장 자원 관리부

확장 자원 관리부의 가장 기본적인 역할은 제어부로부터 온 요청을 각 자원 관리부에 통보하고 그 결과를 받아 일단 MIB에 저장하고 제어부에 결과를 넘기는 것이다. 이 기능은 기존 에이전트가 가지고 있던 기능이다. 하지만 소프트웨어 관리와 관리자의 부담 완화를 위해서 소프트웨어 관리 처리 기능과 자체 폴링 기능, 일부 폴링 기능 등의 일련의 작업을 지시하고 그 결과를 모아서 제어부를 통해 관리자에게 전달하는 기능을 추가로 가지고 있다.

기존 에이전트 기능에 추가된 기능들을 살펴보면 소프트웨어의 상태는 변화가 심하기 때문에 일정 시간마다 하드웨어와 소프트웨어 자원 값을 읽어 이상 유무를 찾아낸다. 이 때 사용되는 일정 시간은 에이전트를 설계할 때 미리 설정되며 이 시간에 맞춰 기능을 수

행하고, 오류가 발생하지 않으면 결과를 MIB에 저장한다. 만약 관리자 request나 에이전트 자체 폴링 요구와 겹칠 수 있으므로 오류 검사를 위해 자원 값을 읽는 경우에는 미리 제어부에 이를 통보하여 겹치는 것을 방지한다. 즉, 확장 자원 관리부에서 수행하는 작업이 제어부에서 올라오는 작업보다 우선 순위를 갖게 되는 것이다.

오류 발생시 Fact Repository에는 (oid, value) 쌍을 넘기는 데, 이때 oid는 오류 발생 관리 객체 식별자이고, value는 해당 관리 객체의 오류 값 또는 상태를 의미한다. 그리고 이 쌍을 EPE(Event Processing Engine)에 오류 발생을 통보하는데도 사용한다. 이후 EPE으로부터 에러 처리를 한 결과를 수신 받아 두 가지로 처리한다. 첫 번째는 오류 발생으로 인해 관련 자원에 대한 설정 값을 변경하여 오류 발생에 대한 조치가 취해질 때까지 해당 자원에 대한 접근을 막는 것이고 두 번째는 오류 처리 결과를 관리자에게 통보하여 관리자의 판단에 도움을 주는 것이다.

3.3.3 SRPP와 PMIT

SRPP(Self Retrieve Polling Part)는 기존 망관리 시스템이 관리자에 의해 일정 시간마다 관리 객체의 값을 검색하여 이를 에이전트 안에 있는 MIB에 저장하는 하는 것에 반해, PMIT에 설정된 시간에 따라 관련 MIB에 대한 값을 확장 자원 관리부를 통해 읽어서 결과를 관리자에게 정기적으로 보고하는 기능을 수행한다.

동작 기능은 폴링 환경 설정 동작과 자체 폴링 동작으로 구별된다. 우선 폴링 환경 설정 동작을 보면, 제어부에서 넘어온 데이터, 즉 (SRPP_OID, value)을 가지고 맞는 PMIT의 설정 값을 변경하게 된다. 이 때 SRPP_OID는 차체 폴링 기능을 위한 시간을 의미하며 value는 해당 폴링 시간에 폴링해야 되는 관리 객체 리스트가 된다. 만약 PMIT에 있는 특정 관리 객체를 삭제하고 싶으면 동일한 관리 객체를 SRPP_OID와 함께 관리자가 전송하게 되며, SRPP에서는 관리자로부터 온 관리 객체 리스트와 PMIT의 관리 객체 리스트를 비교하여 동일한 것이 있으며 해당 관리 객체를 PMIT로부터 삭제한다.

두번째 기능은 자체 폴링 기능이다. 자체 폴링은 PMIT에 있는 폴링 간격 테이블의 내용에 따라 관리 객체들의 값을 일괄적으로 읽어서 이를 에이전트에 있는 MIB에 저장한다. 해당 폴링 시간이 되면 PMIT 설정 변경을 할 수 없게 해놓고 해당 polling interval에 연결된 MO LIST를 읽어 제어부를 통해 폴링 작업을 수행한다. 에이전트에 이런 기능을 부여한 이유는 소프트웨어 관리를 포함하게 되는 에이전트 시스템을 관리자가 전적으로 관리한다는 것은 많은 오버헤드가 있기 때문이다. 이런 단점을 해결하기 위한 방법으로 에이전트에 자체 폴링 기능 기능을 부여함으로써 관리자는 초기 설정만 하면 추가 요청 없이 관련 정보를 에이전트로부터 보고 받을 수 있으며 좀 더 효율적인 에이전트 관리를 가능하게 해 준다. PMIT는 SRPP의 기능을 지원하기 위한

설정 값들을 유지한다.

구성 요소로는 폴링 시간이 같은 관리 객체들의 리스트와 폴링 단위 시간을 가지는 폴링 간격이 있다. PMIT 초기화는 관리자에 의해 수행되는데 SRPP는 관리자로부터 온 SRPP 요청에 따라 PMIT에 있는 설정 값을 변경한다. 관리자는 자신이 원하는 구성 관리를 위해 PMIT의 변수 값을 언제든지 변경할 수 있다. 단, 폴링 간격 종류가 너무 많으면 자체 폴링 동작에서 오버헤드가 발생할 수 있으므로 본 논문에서는 3초, 5초, 1분, 5분, 10분으로 제한했다. 이렇게 함으로써 SRPP를 통해 관리자의 기능 일부를 대신 수행하는 에이전트 모델을 만들 수 있게 된다.

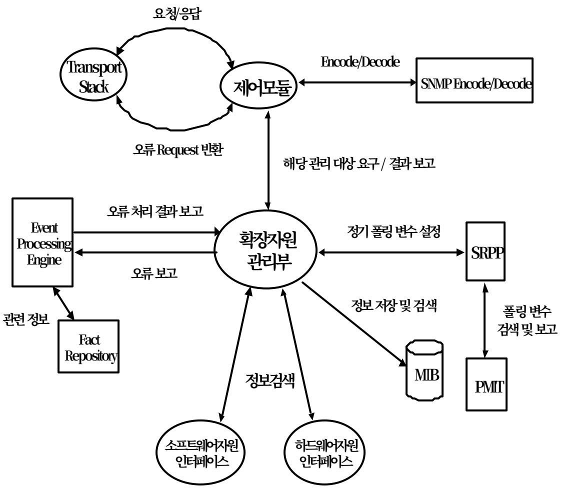
3.3.4 Fact Repository와 EPE

Fact Repository의 주요 기능은 발생 오류에 대한 정보를 저장하고 EPE (Event Processing Engine)에 저장된 기록을 참조할 수 있게 해 준다. 구조와 구성 요소는 Software log table, Hardware log table과 Resource table로 구성된다. 각 테이블은 관련 요소들끼리 연결되어 EPE가 에러 원인을 파악할 때 자료를 제공한다. 그리고 관련 MO list에 대한 포인터를 가지고 있으며 그 다음은 자신의 oid, 과거 발생 원인 list, 자원이나 관련 MO(hardware/software) list 순으로 배열된 형태를 가진다. Software log table인 경우에 각 row는 한 개의 MO와 관련된 사항을 저장하는데, 관련 자원에 대한 포인터를 앞에 놓고 해당 OID, 과거 기록 list, 관련 하드웨어에 대한 포인터가 있다. Hardware log table도 같은 형식으로 구성되며 Resource table은 실질적인 장치에 대한 소프트웨어와 하드웨어 MO들과의 관계를 설정해 놓고 있다. 확장 자원 관리부에서 에러가 발생했음을 EPE에 알리고, 관련 사항을 (oid, value) 형태로 보내오거나 EPE에서 (oid, 원인) 형태로 결과가 Fact Repository에 기록되는 것들을 통해 어떤 MO나 자원에서 에러가 발생하는지와 얼마나 많이 발생하는지를 알 수 있는 일종의 데이터베이스 역할을 한다.

EPE(Event Processing Engine)은 하드웨어와 소프트웨어 자원에 대한 오류가 발생할 경우 확장 자원 관리부로부터 이를 통보 받은 후 Fact Repository로부터 관련 자료를 수집하고 나서 에러 처리 기능을 시작한다. 이 때 Fact Repository에서 참조할 자료는 오류 발생 MO의 oid, 기존 원인 list, 관련 MO list와 연결 자원 list다. EPE의 동작 알고리즘을 살펴보면 우선 에러가 발생한 MO를 통보 받으면 Fact Repository에서 해당 MO의 과거 원인 list를 가져와서 이를 분류한다. 그리고 나서 관련 MO list(오류 발생이 소프트웨어 MO이면 하드웨어 MO)들에 대한 list와 원인 list, 오류 발생 MO와 연관된 자원 list를 가져와서 이미 분류된 원인 list table에 해당 원인과 자원을 분류하여 채워 넣는다. 이 작업은 관련 MO와 자원이 없을 때까지 지속된다. 이 분류 작업을 통해 가장 많이 중복되는 원인과 자원을 오류 발생 요인으로 판단하고 이를 확장 자원 관리부에 (oid,

원인) 형태로 결과를 통보한다.

이와 더불어 Fact Repository에도 결과를 알려 Fact Repository에 등록하게 한다. 이렇게 하고 나서 원인 판단 table을 delete하면 오류 발생에 따른 처리 과정이 끝나게 되며 오류 발생시 hardware와 software 모두를 종합적으로 판단하여 관리자에게 알려 줌으로써 관리자로 하여금 hardware 자원뿐만 아니라 software 자원까지 고려한 망 관리를 할 수 있게 해줄 수 있다. (그림 4)는 하드웨어 자원과 소프트웨어 자원 모두를 관리하기 위해 제시된 소프트웨어 관리 SNMP 에이전트 모델의 전체 동작 흐름을 보여주고 있다.


요청/응답
Transport Stack
제어모듈
Encode/Decode
SNMP Encode/Decode
오류 Request 반환
해당 관리 대상 요구 / 결과 보고
Event Processing Engine
오류 처리 결과 보고
오류 보고
확장자원 관리부
정기 폴링 변수 설정
SRPP
관련 정보
Fact Repository
정보 저장 및 검색
폴링 변수 검색 및 보고
MIB
PMIT
정보검색
소프트웨어자원 인터페이스
하드웨어자원 인터페이스

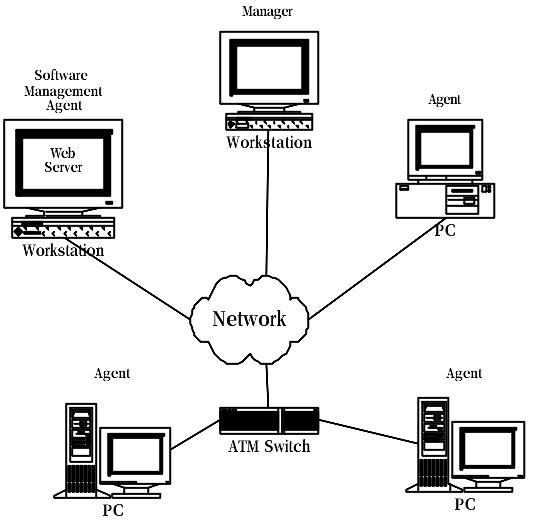
(그림 4) 소프트웨어 관리를 위한 확장 SNMP 에이전트 동작 전체 흐름도

4. 확장 SNMP 에이전트 구현

본 장에서는 3장에서 설계한 확장 SNMP 에이전트를 구현한 결과를 보여주기 위해 구현 환경을 설명하고 정의한 소프트웨어 MIB의 적절한 로딩 여부와 확장 SNMP 에이전트를 구동 결과를 보여준다

4.1 구현 환경

(그림 5)에서 보는 것처럼 테스트 망은 1개의 망 관리자 프로그램과 4개의 SNMP 에이전트를 가지는 5개의 시스템으로 구성하였다. 망 관리자 프로그램과 확장 SNMP 에이전트는 각각 SUN 워크스테이션에 설치하고 나머지3개의 SNMP 에이전트는 데스크탑 컴퓨터에 설치하였다. 망 관리자 프로그램은 망을 구성할 때 관리 영역에 있는 모든 노드의 상태를 가져와 현재 각 노드의 상태를 윈도우에 표시해 준다.


Manager
Workstation
Software Management Agent
Web Server
Workstation
Agent
PC
Network
Agent
PC
ATM Switch
Agent
PC

(그림 5) 구현 환경

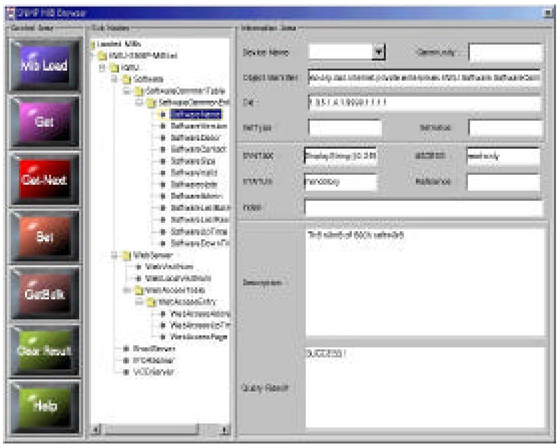
확장 SNMP 에이전트는 UCD SNMP의 에이전트 프로그램을 기본 에이전트 프로그램으로 하고 본 논문에서 제시한 기능들을 지원하는 모듈들을 추가하여 만들었다. Apache Server 프로그램에

3.1장에서 정의한 소프트웨어 MIB을 적용시켰다.

4.2 소프트웨어 MIB 구현

소프트웨어 MIB는 SMI로 정의되며 관리자 프로그램과 에이전트가 초기에 실행될 때 각각 MIB를 자신의 메모리에 로드 한다. 적절하게 로드 되었는지를 확인하는 방법 중에 하나는 MIB Browser를 사용하는 것이다. (그림 6)은 SNMP MIB Browser를 사용하여 3장에서 정의한 소프트웨어 MIB를 브라우징한 결과를 보여주고 있다.

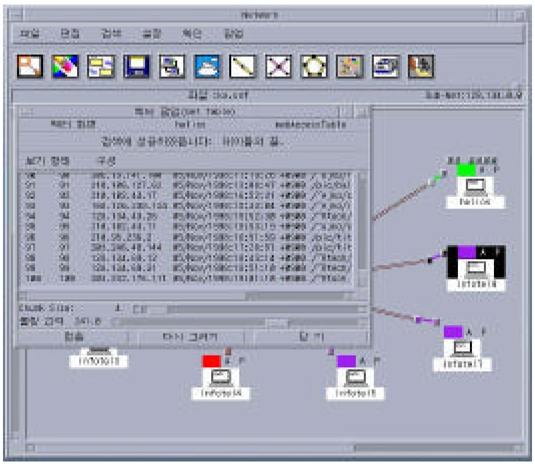
계층적으로 각 관리 객체들이 MIB Browser 왼쪽에 보여지고 있으며 오른쪽에는 선택된 관리 객체의 OID를 비롯하여 SMI에 의해 정의된 사항들을 보여주고 있다. 오른쪽 맨 하단에 있는 Query Result는 MIB 로드의 성공 유무를 나타낸다.


Mib Load
Get
Get-Next
Set
GetBulk
Clear Result
Help
Device Name
Community
Object Identifier
Oid
Description
Query Result

(그림 6) 소프트웨어 MIB 구현

4.3 소프트웨어 에이전트 구현

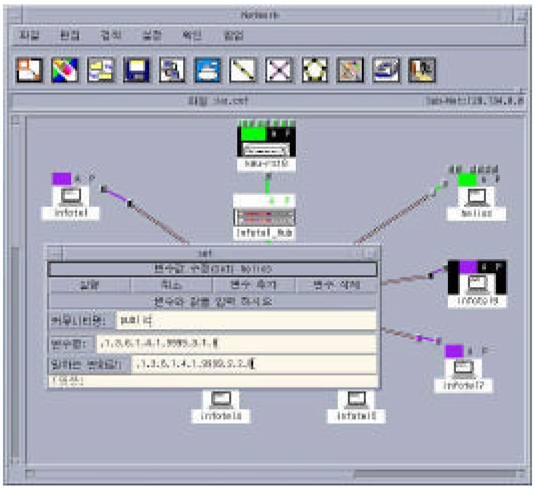
에이전트 구현에서 가장 기본적인 것은 관리자가 원하는 관리 대상 객체의 정보를 망 관리 에이전트가 적절하게 추출하여 망 관리자 프로그램에 전달하는 것이다. 이 기능을 검사하기 위해 망 관리자 프로그램에서 제공하는 노드 관리 정보 추출 기능을 사용하였다. (그림 7)은 소프트웨어 관리 에이전트가 있는 노드를 마우스로 클릭하여 확장 SNMP 에이전트가 관리하고 있는 웹 서버의 검색 페이지 리스트라는 관리 정보를 선택한 후 정보 요청을 한 후, 망 관리자 프로그램의 윈도우에 해당 관리 정보가 표시하고 있음을 보여 주고 있다.



(그림 7) 소프트웨어 MIB 추출 결과

(그림 8)은 관리자와 에이전트간 망 관리 데이터 트래픽을 줄이기 위해 확장 SNMP 에이전트가 제공하는 self-polling 기능 구현 화면이다. 확장 SNMP 에이전트에 있는 PMIT의 값을 설정하기 위한 명령 모드를 보여주고 있다. 이 명령을 통해 에이전트는 망 관리자 프로그램의 요청이 없어도 미리 설정된 시간에 따라 자신의 망관리 정

보들중 관리자가 미리 정의한 정보만을 추출하여 망 관리자에게 보고하게 된다.



(그림 8) 관리자에 의한 PMIT 설정

5. 결 론

망이 다양해지고 복잡하게 발전함에 따라 하드웨어 요소뿐만 아니라 각 노드에서 동작중인 주요 소프트웨어 즉, 미들웨어나 서버 프로그램들의 비중이 증가하면서 망 성능에 소프트웨어의 영향이 커지고 있다. 따라서 이들 소프트웨어들에 대한 관리 필요성과 기존 망 관리와의 일관성 유지 문제가 제기되고 있다.

또한, 망의 규모가 커짐에 따라 관리 기능의 대부분을 수행하는 관리자의 부담이 증가되고, 각 에이전트와의 관리 정보 전송을 위한 트래픽도 증가하고 있다.

이런 문제점들을 해결하기 위해 본 논문에서는 각 도메인별로 해당 노드에 있는 주요 소프트웨어 관리기능을 가지면서 관리자의 고유 기능 중 폴링과 에러 검출을 수행하기 위한 기능들을 가지는 확장 SNMP 에이전트 모델을 제시하고 소프트웨어 공통 부분, 웹 서버, 폴링, 메일 서버, IPOAARP 서버, VOD 서버에 대한 MIB를 정의하였다. 이들 중 웹 서버 프로그램을 대상으로 확장 SNMP 에이전트 모델을 구현하였다.

확장 SNMP 에이전트는 관리자의 요구에 따라 소프트웨어를 포함한 관리 대상 객체의 값을 추출하여 전달하고 관리자에 의해 미리 설정된 시간에 따라 자신이 관리하고 있는 망 관리 정보를 관리자에게 보낸다.

이런 과정을 통해 관리자는 자신이 관리하는 각 에이전트의 하드웨어 상태뿐만 아니라 해당 노드에 있는 주요 소프트웨어의 상태를 모두 고려하여 보다 효율적인 망 관리 기능을 제공해 준다. 본 논문에서 제시한 확장 SNMP 에이전트의 폴링과 에러 검출 기능으로 인해 관리자의 부담이 감소하고 원하는 관리 객체에 대한 정확한 값을 알게 되었다. 폴링후 관리 객체의 값이 변경된 것만을 관리자에게 알려주거나 에러 원인을 알려 줌으로써 관리자와 에이전트간의 관리 정보 전송에 따른 트래픽도 감소하였다.

앞으로 연구 진행 방향은 많은 프로그램에 본 논문에서 제시한 에이전트 모델을 적용할 수 있게 해주는 작업과 성격이 다른 프로그램들간의 공통적인 관리 객체를 표준으로 정의하고 이와 별도로 개별 특성을 특정 관리 객체로 체계적으로 분류하는 일이다. 또한 효율적인 원인 검색 방법에 대한 연구도 진행되어야 할 것이다.

[참고 문헌]

[1] Brian Keegan, http://www.cifs.com/2ndcifconf/SCO-Keegan

[2] C. Kalbfleisch, C. Krupczak, R. Presuhn, J. Saperia, "Application Management MIB", IETF Draft, 1997

[3] C. Krupczak, J. Saperia, "Definitions of System-Level Managed Objects for Applications", RFC 2287, 1998

[4] David Perkins, Evan Mcginnis, "Understanding SNMP MIBs", Prentice Hall PTR, 1997

[5] Divakara K. Udupa, "Network Management Systems Essentials", McGraw-Hill, 1996

[6] Doug Steedman, http://www.techapp.co.uk/asn1gloss.html

[7] Iosif G. Ghetie, "Networks And Systems Management", Kluwer Academic Publishers, 1997

[8] ITU-T, "Systems management: Software management function", X.744, 1996

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

[10] Marshall T. Rose, "The Simple Book", Prentice Hall PTR, 1996

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

[12] Sidnie M. Feit, "SNMP", McGraw-Hill, Inc., 1995

[13] William Stallings, "SNMP, SNMPv2 and CMIP", Addison Wesley, 1993

고은진

1997 광운대학교, 전산학 학사

1999 광운대학교, 전산학 석사

1999~현재 ETRI 연구원

<관심분야> 망관리, Active Network

이길행

1986 KAIST 전산학과 석사

1996 KAIST 전산학과 박사

1986~현재 ETRI 책임연구원

(네트워크기술연구소, 네트워크서비스 연구부, 통합망관리팀장)

<관심분야> Distributed Processing, Load Balancing, Network Management, DBMS

조국현

1977 한양대학교 전자공학 학사

1981 Tohoku University, 공학석사

1984 Tohoku University, 공학박사

1984~현재 광운대학교 컴퓨터과학과 정교수

<관심분야> 망관리, Active Network, IPv6