네트워크 관리 모델에서의 이동 에이전트 패러다임

(Mobile Agent Paradigm for a Network Management Model)

최 원 상 † 김 태 윤 ††
(Won-Sang Choi) (Tai-Yun Kim)

요 약 현재 널리 사용되고 있는 네트워크 관리 기술들은 서로 다른 관리 정보 모델을 정의하여 사용하거나 다른 관리 프로토콜을 사용하여 통신하므로 호환성을 가지지 못한다. 이런 문제를 해결하기 위해서 여러 분산 패러다임들의 적용을 통해 이전 모델의 개선을 위한 연구들이 이루어지고 있다. 그러나 기존의 연구들은 그 모델의 효용성을 양적으로 제시하기보다는 질적이고 사고 중심적으로 제시하였다. 그래서, 본 논문에서는 모빌 에이전트 패러다임을 이용한 네트워크 관리 모델을 제안하고 그 적용의 양적인 실효성을 알아본다. 그 실효성의 양적 비교를 위해 여러 분산 패러다임과 새로이 제안하는 모델간의 효용성을 각 모델의 통신량 패러메터 모델을 통한 통신량 크기 비교를 통해서 제시한다.

Abstract Traditional network management technologies don't have interoperability to use different protocols or management information models each other. Many researchers have tried to find solutions of these problems to use distributed paradigms. But the benefits of existing models are mainly supported only by qualitative evidences rather than by quantitative evidences. In this paper, we present a quantitative evidence of the efficiency of network management model using mobile agent paradigm. To compare distributed paradigms and proposed model, we use parameterized traffic models for measuring the amount of whole traffic generated by each model.

1. 서 론

네트워크 관리(Network Management)란 통신망을 효율적으로 운용, 유지, 보수하기 위한 정보 통신 기술을 말한다. 이와 같은 통신망 관리의 상호 운용성을 제공하기 위해서 ISO/OSI는 CIMP(Common Management Information Protocol)를, IETF에서도 TCP/IP Internet의 관리를 위해 SNMP(Simple Network Management Protocol) 표준을 개발하였다. 그러나, 대부분의 현재 사용중인 모델은 매니저/에이전트(Manage/Agent) 구조에 근거한 중앙 집중형의 클라이언트/서버(Client/Sever) 형태를 취하고 있고, 그 사용에 대한 관리자의 방대한 경험과 지식이 필요하고 관리 정보 설정의 지역적 한계가 있다. 더구나 현재 널리 사용되고 있는 기술들은 서로 다른 관리 정보 모델을 정의하여 사용하거나 다른 관리 프로토콜을 사용하여 통신하므로 호환성을 가지지 못한다[1]. 따라서 관리자들은 자신이 관리하는 시스템이나 네트워크에서 지원하는 관리 기술에 따라 여러 가지 관리 도구들을 혼용해야만 하고, 이것은 관리자에게 큰 부담이 되고 있다.

그래서 이와 같은 문제를 해결하기 위해서 여러 분산 패러다임들의 적용을 통해 이전 모델의 개선을 위한 연구들이 이루어지고 있다. 현재 활발히 이루어지고 있는 분산 모델은 크게 두 가지 방향으로 잡혀져 있다. 첫 번째는 강력한 분산 관리 패러다임을 사용하여 기존의 패러다임의 약점인 상호운용성을 해결하고, MbD (Management by Delegation) 모델에 기반하여 크기에 제한 받지 않고, 유연성과 확고성을 제공하는 것이 있다[2]. 두 번째는 소프트웨어 엔지니어링 관점에서 이미 고안된 CMIP와 SNMP에 새로운 분산 오브젝트 아키

· 이 논문은 1997년 한국학술진흥재단의 학술연구비에 의하여 지원되었음
† 학생회원 : 고려대학교 컴퓨터학과
lilith@netlab.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
tykim@netlab.korea.ac.kr
논문접수 : 1999년 5월 16일
심사완료 : 1999년 11월 17일

텍처를 사용하여 수정하는 것으로 현재 CORBA 아키텍처를 이용한 분산 객체 모델로의 확장에 대한 연구가 활발히 진행되고 있다.

본 논문에서는 이동 에이전트 패러다임을 이용한 네트워크 관리 모델을 제안하고 그 적용의 실효성을 알아본다. 제안하는 모델은 자바에 기반한 이동 에이전트 프레임워크인 Aglet을 이용해서 설계되어 상호운용성을 제공하고, MbD 모델에 기초하여 설계되었기 때문에 크기에 제한 받지 않으며, 유연성과 확고성을 제공한다. 또한 Aglet[3]은 MAF(Mobile Agents Facility)에 기초한 분산 오브젝트 프레임워크이므로 CMIP와 SNMP와 같은 기존의 시스템과의 운용에 필요한 인터페이스를 정의하여 기존 시스템과의 연결도 도모할 수 있다.

기존의 연구들이 그 모델의 효용성을 질적이고, 피상적인 사고에 의한 것으로 기술하고, 양적인 증거나 해결책에 대한 자세한 분석이 부족했던 것과 달리 본 연구에서는 여러 분산 패러다임과 새로이 제안하는 모델간의 효용성을 양적 비교를 위해서 각 모델의 통신량 패러메터들간의 크기 비교를 통해서 알아본다.

본 논문의 구성은 다음과 같다. 2장에서 네트워크 관리 패러다임들에 대해 살펴보고, 3장에서 제안하는 이동 에이전트 패러다임 모델에 대한 설계와 관리 시나리오와 기존의 MbD 모델과 비교를 기술하고, 4장에서 각 모델의 통신란 패러메터 모델에 기반한 사례 연구를 통해 통신량 비교를 통한 수행 성능을 평가하고, 5장에서 결론 및 향후 연구 방향을 제시한다.

2. 네트워크 관리 패러다임

네트워크 관리 패러다임은 크게 고전적인 모델인 중앙집중형 패러다임과 분산 패러다임으로 나눌 수 있고, 다시 분산 패러다임은 연약한 분산 패러다임과 강력한 분산 패러다임으로 나뉘어진다[4]. 이번 장에서는 각 패러다임에 대해서 간략히 알아본다.

2.1 중앙 집중형 패러다임

중앙 집중형 패러다임이란 보통 클라이언트/서버의 구조를 가진 전통적인 모델이 이에 속하는데, 이런 모델로는 SNMP, CMIP가 있다.

2.1.1 SNMP[5]

SNMP는 기본적으로 3가지 동작으로 네트워크를 관리한다.

(1) GET: 관리자가 대리인에 있는 객체의 값을 가져온다.

(2) SET: 관리자가 대리인에 있는 객체의 값을 변경한다.

(3) TRAP: 대리인 특정 상황 발생을 관리자에게 알린다.

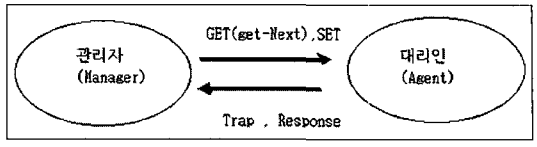

그림 1 SNMP의 기본 통신구조

그림 1과 같이 SNMP는 관리자가 물음에 대리인이 응답하는 것이 기본이고 대리인은 특수한 상황이 발생한 경우에 관리자에게 발생을 알리고 필요한 정보를 함께 보낸다. SNMP version 1에는 5가지의 PDU(protocol data unit), 기능이 정해져 있다. 그림 2의 "PDU type"이 아래의 5가지중 하나이다.

(1) GetRequest. 관리자가 특정 객체의 한 값(object instance)을 읽어올 수 있게 한다.

(2) GetNextRequest. 관리자가 지정한 객체 값의 다음 객체 값이나 지정한 객체가 테이블인 경우 다음 index의 값을 가져올 수 있게 한다.

(3) GetResponse. 관리자의 요구에 대리인이 해당 객체 값을 돌려준다.

(4) SetReuest. 관리자가 대리인에게 있는 객체 값을 변경한다.

(5) Trap. 대리인이 특정 상황이 발생했음을 관리자에게 알린다.

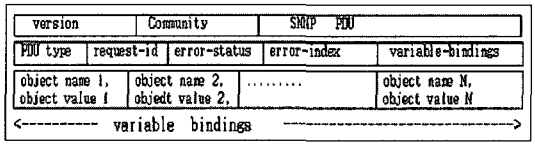

그림 2 SNMP Message Format

SNMP에 의해서 통신을 이루기 위해서는 그림 2와 같은 형태의 메시지를 만들어야 한다. 이때 통신을 하기 위해서 다음의 사항들은 반드시 있어야한다.

(1) Version. 관리자와 대리인 시스템간의 SNMP version이 일치해야한다.

(2) Community. 양 시스템간의 Community name이 일치해야 한다.
이것은 최소한의 인증 절차를 위한 암호기능을 한다.

(3) PDU type. 관리자가 요청하는 경우에

GET/GetNext, SET 이외의 PDU type이 되면 응답이 없다. 마찬가지로 GET/GetNext에 대한 응답 시에 GetResponse가 아닌 PDU type이 오면 해석이 불가능하다.

2.1.2 CMIP[6]

CMIP는 잘 정리되고 훌륭한 개념임에도 불구하고 지나치게 방대하게 규정되어 있어서 구현하기가 쉽지 않고 시스템도 SNMP에 비해서 무척 크다. 그래서 쉽게 SNMP를 대체하지 못하고 있다. 그러나 최근 TMN (Telecomunication Management Network)이라는 개념이 도입되면서 CMIP/CMIS가 주목을 받고 있고, CMIP를 이용한 각종 개발 도구들이 발표되고 있다. 이에 대한 자세한 내용은 뒷장에 기술한다.

2.2 분산 패러다임 모델

분산 패러다임이 네트워크 관리에 요구되고 있는 것은 현재 사용되고 있는 모델이 호환성 이 부족하고 네트워크 환경이 대규모 분산 환경으로 바뀌고 있기 때문이기도 하지만, 그에 못지 않게 현재 사용중인 기본 에이전트의 성능이 부족한 것도 한 원인이다. 현재의 기본 에이전트는 네트워크 장치에 있는 제한된 자원을 사용하는 데 있어서 그 사용량이 많기 때문에 적은 자원을 사용하는 에이전트가 요구되고, 에이전트의 동적인 주문에 의한 생성이 가능해야 한다. 그래서 이번 장에서는 여러 분산 패러다임의 특징에 대해 살펴본다.

2.2.1 연약한 분산 패러다임

1) RMON(Remote network-MONitoring)[7]

네트워크의 효율적 이용을 위해서 현재의 네트워크 상태를 측정하고 과거의 기록을 토대로 앞으로의 네트워크 문제를 사전에 예견하고 제거하기 위해서 유용하게 이용되는 것이 RMON(Remote network-MONitoring)이다. 중앙에서 원거리 지역의 네트워크의 전체 이용현황을 적은 대역폭으로 쉽게 알 수 있게 해준다. 기존의 SNMP MIB들이 에이전트가 탑재된 장비 자신의 처리 결과만을 보유하고 있는 데 반해서 RMON 에이전트는 한 세그먼트 전체에서 발생하는 트래픽을 파악하게 해준다. 즉 전체 발생 트래픽, 세그먼트에 연결된 각 호스트의 트래픽, 호스트들 간의 트래픽 발생현황을 알려준다. 이런 처리를 위해서 RMON 에이전트는 전체 통계 데이타, 이력 데이타(history), HOST 관련 데이타, HOST MATRIX와 사전에 문제 예측 및 제거를 위해서 특정 패킷을 필터링 하는 기능과 임계치를 설정해서 이에 도달하면 자동으로 알려주는 경보기능 및 사건 발생 기능을 보유한다. 에이전트의 결과는 RMON 프로브(probe)가 세그먼트 전체의 상황을 모아서 통보해준다.

2) OSI 관리 프레임워크와 TMN

전술했던 대로 ITU-T에서 OSI 관리 프레임워크를 TMN model의 기초로써 사용되고 있다. CMIP은 관리 정보를 전송하는 절차 즉 CMISE(Common Management Information Service Element)사이에 CMIS (Common Management Information Service)를 완성시키기 위해서 교환하는 CMIP PDU(Protocol Data Units) 만들고 전송하는 것에 대해 정의해 놓은 것이다. 양단의 CMISE 사용자(관리자와 대리인 또는 양단의 CMISE 어플리케이션)들이 정보교환을 위해서 시스템을 연결하는 데 ACSE(Association Control Service Element)를 이용하는 데 이때는 CMIP이 이용되지 않는다. 관리 서비스를 위해서 CMISE는 PDU들을 교환하기 위해서 CMIP를 채용한다. 그리고 CMIP는 CMIP PDU 전송을 위해서 ROSE(Remote Operations Service Element)를 이용한다.

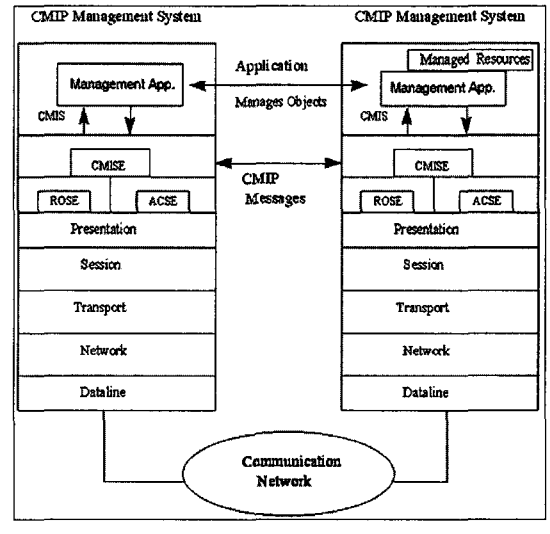

그림 3 OSI Management 구조

CMIP는 CMIS 서비스를 처리하기 위해서 11개의 PDU를 정의해놓았다. ROSE는 몇 가지 Association class를 정해 놓고 있다. CMIP는 항상 ROSE 3을 이용한다. 그리고 확인 서비스(Confirmed CMIS)를 위해서는 class 2 또는 3을 이용하고 비 확인 서비스를 위해서 class 5를 이용한다.

2.2.2 강력한 분산 패러다임

1) MbD (Management by Delegation)

MbD 모델은 네트워크 관리를 네트워크 관리 서버에

서 하는 것이 아니라 유연성을 제공하는 서버(MbD server)가 관리 서버를 대행하여 관리 오퍼레이션을 동적으로 분산하여 처리하는 모델을 말한다. MbD 서버는 주문형의 네트워크 관리를 제공하고, 관리 수단과의 효과적인 연결을 제공하며, 기존의 관리 기법과의 호환성을 제공한다.

2) 이동 코드(Mobile Code)

Fuggetta et al.는 이동 코드에 대한 자세한 조사를 하였다. 그는 명확하게 MCS(Mobile Code System)를 다음의 세 가지 형태로 나누었다[8]. 세 가지 형태의 특징은 다음 표 1과 같다. 표 1에서 대문자로 표기된 A, B는 그 시스템에서의 상태를 나타내고, 이텔릭체로 표기한 것은 이동한 것을 의미한다. 그리고 밑줄이 그어진 시스템에서 코드가 실행된다.

표 1 이동 코드 패러다임의 특징

Paradigm	Before moving		After moving	
	System A	System B	System A	System B
REV	Know-how A	Resource B	A	*Know-how* Resource **B**
COD	Resource A	Know-how B	Resource *Know-how* **A**	B
MA	Know-how A	Resource	-	*Know-how* Resource **A**

(1) Remote EValuation(REV)

클라이언트가 서버에 있는 서비스를 호출할 때, 서비스의 이름과 입력 패러메터들뿐만 아니라 코드도 같이 보내는 형태이다. 그래서 이 패러다임은 클라이언트가 서비스를 수행하기 위한 코드를 소유하는 반면에 서버는 클라이언트에서 보내오는 코드를 수행할 자원과 환경을 소유한다. REV는 RPC(Remote Procedure Call)의 확장으로 볼 수 있다.

(2) Code On Demand(COD)

클라이언트가 주어진 목적의 일을 수행해야할 때, 코드 서버에 접속해서 서버로부터 필요한 코드를 내려받고, 동적인 코드 바인딩을 통해서 그것을 수행하는 것이다. 그래서 클라이언트는 자원을 소유하고, 서버는 코드를 소유하는 형태의 패러다임이다.

(3) Mobile Agent(MA)

이동 에이전트는 자율적으로 하나의 호스트에서 다른 호스트로 전이해 가면서 끊김 없이 실행을 해 가는 형태의 패러다임이다. 개념적으로는 이동 에이전트는 호스트간 이동에서 가상 기계 전체를 전이할 수 있다. 이 패러다임에서는 이동 에이전트는 코드를 소유하고 호스트가 자원을 소유한다.

2) 분산 오브젝트(Distributed Object)

(1) JMAPI

JMAPI는 시스템과 네트워크 어플리케이션을 통합하여 원격지 정보를 관리하기 위해 여러 가지 통합하여 원격지 정보를 관리하기 위해. 여러 가지 어플리케이션을 사용해야 했던 문제점을 해결하기 위해 개발되었다. 이전에 있던 여러 가지 다른 종류의 자율 어플리케이션들을 어플리케이션 단위로 통합함으로써 실제적으로 어플리케이션이 수행될 수 있었던 모든 환경에서 함께 수행될 수 있게 되었다. 이러한 특성은 관리 작업을 단순하게 해준다. 결국 JMAPI는 개발자가 자바 컴퓨팅 환경의 장점을 이용함으로써 통합 관리 솔루션을 제공하는 툴을 더욱 쉽게 개발할 수 있게 하기 위해 SUN에서 개발한 자바 확장 클래스이다. 이 클래스들은 이질적인 네트워크 환경 상에서의 시스템, 네트워크, 그리고 서비스 관리를 위한 프로그램개발을 위해 확장된 객체와 메소드 들의 집합이다.[1] JMAPI는 MIB를 Script화된 MIB로 바꾸어서 사용하며, 그 관리 함수의 호출은 RMI를 통해서 이루어지며, HTTP를 통해서 그 관리 정보를 보여준다.

(2) CORBA

JIDM(Joint Inter-Domain Management) 그룹과 NM(Network Management) 포럼은 CORBA와 CMIP, SNMP 기반의 시스템들과의 통합을 위한 툴을 모색했다. 그래서 NM 포럼의 IIMC(ISO-Internet Management Coexistence) 그룹은 SNMP와 CMIP 서비스, 프로토콜과 정보를 번역하는 상세화 작업을 하였다. 그리고 JIDM은 CMIP/CORBA, SNMP/CORBA를 모두 다루는 GDMO/ASN.1 과 CORBA IDL 그리고 SNMP MIB와 CORBA IDL을 매핑하는 알고리즘을 정의하였다. 이를 통해서 프로그래머는 IDL 기반의 자원, 서비스를 처리할 수 있게 되었고, IDL을 알지 못하는 어플리케이션의 처리, 반대로 GDMO, ASN.1과 SNMP, CMIP를 알지 못하는 관리자와 에이전트에 관한 처리를 할 수 있게 되었다.

(3) WBEM

1996년 7월에 마이크로소프트, 인텔, BMC 소프트웨어, 시스코, 컴팩 등이 웹 기반 관리를 실제 산업체에 적용시키기 위해서 만든 컨소시엄이다. 그 목적은 웹 기반 관리의 장점을 전 산업체에 파급시키는 것으로 하고

있으며, 그에 따라 웹 브라우저로 관리할 수 있는 구조로, SNMP, HTTP, CMIP, DMI 등의 현재 사용되고 있는 관리 기술을 모두 수용하고, 현존하는 산업 표준을 지원하기 위한 방안을 마련하고자 하며, 또한 관리 솔루션을 통합하기 위한 공통 기반을 제공함으로써 모든 관리 정보를 표현하는 공통의 정보 모델을 제정하려고 할 뿐 아니라 웹과 인터넷 기술의 장점인 널리 퍼져있고 개방적이다라는 특성을 활용하여 이를 관리 행위에 이용하려고 하고 있다. 이런 노력의 결과로 WBEM은 HMMS(HyperMedia Management Schema)와 이러한 형식으로 표현된 관리 정보를 주고받기 위한 프로토콜인 HMMP(HyperMedia Management Protocol)와 WBEM을 준수하는 어플리케이션이 네트워크 구성요소를 하나의 객체로 관리할 수 있도록 해주는 소프트웨어 개발 기술인 HMOM(HyperMedia Object Manager)을 표준안이 있다.

(4) ODMA

ODMA의 목적은 OSI 관리 아키텍처를 RM-ODP(Reference Model of the ISO Open Distributed Processing) 프레임워크를 사용하여 확장시켜서 이종의 큰 크기의 분산 시스템인 TMN 아키텍처로 확장시키는 것이다. 기본적으로 객체 지향 분산 관리 아키텍처이며, 계산 객체(computational object)들로 구성된다. ODMA에는 OSI 관리 프레임워크와 같이 더 이상 관리자나 에이전트에 고정된 역할이 없다. 대신 계산 객체들이 관리 역할과 에이전트의 역할에 대한 인터페이스를 제공할 수 있다. 더구나 ODP, ODMA는 관리 어플리케이션에 투명하게 위치를 알려줄 수 있다. 그러므로, 에이전트는 NMS와 같은 수준의 더 향상된 관리 목적을 수행할 수 있다. 결국, ISO와 ITU-T는 OSI 관리 프레임워크의 연약한 분산 관리 패러다임에서 ODMA를 통해 강력한 분산 관리 패러다임으로 변화를 꾀하고 있다.

3. 제안하는 이동 에이전트 패러다임 모델

강력한 분산 패러다임들은 위와 같이 나눌 수는 있지만, 실제 적용에 있어서는 여러 가지 패러다임들이 서로 연관되어 사용되고 있다. 예를 들어 JMAPI의 경우에는 분산 오브젝트 패러다임의 일종이지만, 내부의 구조를 살펴보면 JAVA의 RMI를 사용하여 REV 형태의 이동 코드를 사용하고 있으며, 본 논문에서 제안하는 이동 에이전트 패러다임 모델에서도 이동 코드 패러다임인 이동 에이전트를 사용하면서 그 기반 프레임워크를 CORBA기반의 Aglet을 사용하고 있다. 그리고, 현재 자바 코드를 현재 상용 에이전트에서 직접 처리할 수 없으므로, 이를 위한 게이트웨이를 통한 대행자 방식을 사용한다. 그러므로, 이동 에이전트는 상용 에이전트에서 기동되는 것이 아니라 게이트웨이를 제어하는 CORBA 기반의 매니저에서 처리되어 실행된다. 이동 에이전트 기반 네트워크 관리의 장점으로는 이동 에이전트가 필요한 망 노드(Node)에서만 실행되므로 효율성 향상, 자원사용량 감소되고, 이로 인한 망 사용량 감소, 비동기화 방식의 통신, 온라인 서비스 향상 및 확장성 제공하는 것이 있다[9].

3.1 전체적인 모델 구조

그림 4는 이동 에이전트 모델의 전체 구성도이다. 이 모델의 주요 구성요소는 다음과 같다.

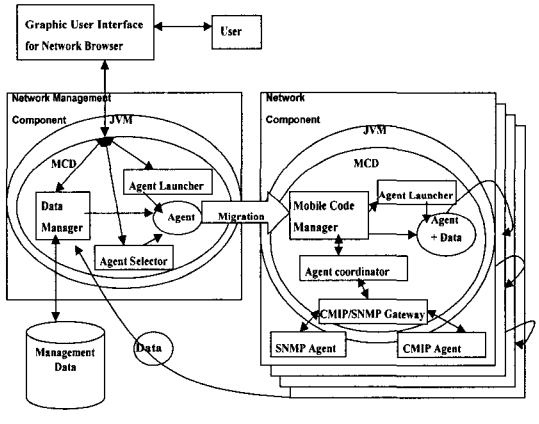

그림 4 전체 구성도

- 주 요소

◆ 그래픽 사용자 인터페이스(Graphic User Interface): 사용자가 네트워크 상황 파악과 관리를 위한 인터페이스로 사용자가 원하는 이동 에이전트를 쉽게 선택할 수 있도록 하였다. 다음 그림 5는 구현된 사용자 인터페이스이다. 사용자는 원하는 에이전트를 선택해서 생성시킬 수 있다.

◆ 네트워크 관리 컴포넌트(Network Management Component): 사용자가 브라우저를 통해 보낸 관리 명령에 따른 관리를 처리하는 모듈로 인터페이스로부터 입력된 정보를 바탕으로 이동에이전트 코드를 생성하고, 그 코드가 가져오는 데이타를 처리한다.

◆ 네트워크 컴포넌트(Network Component): 실제로 네트워크 관리가 이루어지는 곳으로, 이동 에이전트가 옮겨져서 실행이 이루어지는 네트워크 관리 서버.

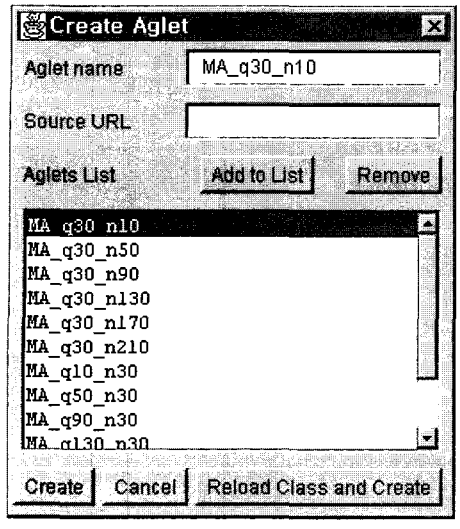

그림 5 에이전트 생성 인터페이스

- 공통 요소

◆ 이동 코드 데몬(Mobile Code Daemon): 이동 코드 데몬은 이동 에이전트 프레임워크인 Aglet의 서버 데몬(Server Deamon)에 해당되며, 이동 에이전트인 Aglet을 모니터링하고, 도착되는 에이전트를 받고, 에이전트 컨텍스트(Context)를 제공하는 이동 코드 관리자에 에이전트를 보내는 역할을 한다.

◆ 자바 가상 기계(Java Virtual Machine): 이동 코드 데몬이 작동하는 자바 환경으로, 에이전트와 에이전트 서버, 에이전트 관리자 등의 수행에 필요한 환경을 제공한다.

◆ 이동 에이전트(Mobile Agent):네트워크 관리를 위한 이동 코드로 실행 코드와 데이타를 수용한다. 수용되는 실행 코드와 데이타는 아래 표 2와 같다.

표 2 이동 에이전트 수용 데이타

Name	Type
Agent ID	32-bit bit-stream
Launching Management Component Addr.	String
Itinerary Policy	String
SNMP Instructions + MIB variables + MIB variable value	Management Instruction Class
Network Component Addr. List	String Vector

* 이동 에이전트의 순회 스케줄 정책(Itinerary Policy)

기본 순회 정책(Simple Itinerary Policy): 네트워크 컴포넌트 주소 리스트에 있는 정적으로 정해진 순서대로 순회하는 정책으로 Dispatch에 의해 에이전트를 전송한다.

다중 순회 정책(Multi Choice Itinerary Policy): 네트워크 컴포넌트의 에이전트 이송자에서 다음 순회 네트워크 컴포넌트 경로를 결정해주는 방식으로 에이전트의 전송을 이송자에서 신뢰성 있게 담당한다. 에이전트의 Clone을 생성하여 Dispatch한다.

철회 순회 정책 옵션(Wait for retract Itinerary Policy Option): 위의 두 가지 정책에 덧붙여서 사용자가 언제든지 순회 중에 에이전트를 철회(Retract)할 수 있도록 하는 옵션 정책으로 철회 메시지를 이동 코드 관리자가 받을 수 있도록 한다.

- 망 관리 구성요소

◆ 데이타 관리자(Data manager): 사용자가 보낸 정보를 바탕으로 이동 에이전트를 생성하고, 이동 에이전트가 수집한 정보를 저장하고, 사용자가 브라우저를 통해 그 정보를 볼 수 있도록 하는 모듈이다. 사용자가 철회를 원할 경우 철회 메시지를 네트워크 컴포넌트 주소 리스트(Network Component Addr. List)에 있는 네트워크 컴포넌트들에 보내는 역할을 담당한다.

◆ 에이전트 선택자(Agent Selector): 사용자가 원하는 요구사항에 해당되는 에이전트를 선택하는 모듈이다. 데이타 관리자에 의해 이미 만들어져 있는 이동 에이전트가 사용자의 요구사항과 일치할 경우에 선택해서 바로 처리한다. 이동 에이전트의 캐쉬에 해당하는 부분이다. 에이전트는 에이전트 ID를 통해서 관리된다.

◆ 에이전트 이송자(Agent Launcher): 선택자에 의해 선택된 에이전트를 외부 망 요소로 보내는 모듈로 이동 에이전트가 실행될 서버의 주소를 에이전트에 추가하며, 네트워크 컴포넌트 주소 리스트를 이동 코드 관리자로부터 받아서 관리한다.

- 망 요소 구성요소

◆ 이동 코드 관리자(Mobile Code Manager): 이동 에이전트에 대한 처리를 담당하는 요소로서 이동 에이전트의 실행과 데이타 처리를 담당한다. 에이전트 컨텍스트를 제공하여 이 곳에서 에이전트 Proxy를 생성하여 에이전트를 처리한다. 에이전트 컨텍스트는 장소(Place)에 해당된다. 에이전트에서

SNMP Instructions + MIB variables + MIB variable value에 대한 정보를 에이전트 조정자로 보내고, 그에 대한 답신을 받아서 에이전트에 정보를 추가한다. 또한 에이전트의 순회 스케줄 정책을 처리하며, 에이전트 이송자에게 에이전트의 네트워크 컴포넌트 주소 리스트 중에서 아직 방문하지 않은 리스트를 보내어 에이전트를 다음 네트워크 컴포넌트로 보내게 한다.

- ◆ 에이전트 이송자(Agent Launcher): 선택자에 의해 선택된 에이전트를 외부 망 요소로 보내는 모듈로 이동 에이전트가 실행될 서버의 주소를 에이전트에 추가하며, Network Component Addr. List를 이동 코드 관리자로부터 받아서 관리한다.
- ◆ 에이전트 조정자(Agent Coordinator): SNMP/CMIP 에이전트에 접근을 처리해주는 모듈로 SNMP Instructions + MIB variables + MIB variable value를 해당하는 데이타로 매핑하고, 그 결과 값을 MIB variable value에 업데이트하도록 이동 코드 관리자에게 넘긴다.
- ◆ SNMP/CMIP 게이트웨이(Gateway): SNMP/CMIP 에이전트에 대한 게이트웨이로 에이전트 조정자에서 받은 데이타를 각 해당되는 프로토콜로 관리 장치에 전송하고 결과 값을 받아서 에이전트 조정자에게 보낸다.

3.2 관리 시나리오 및 기본 설계

3.2.1 에이전트 이동 시나리오

에이전트의 이동은 연약한 이동만을 지원한다. 즉, 에이전트가 이동할 때, 데이타의 상태만을 유지하여 보내는 정책으로 이동을 지원한다. 또한 Aglet 프레임워크에서 지원하는 Code_Base를 이용하여 필요한 Class를 동적으로 보내 사용할 수 있다.

기본적인 이동 형태는 다음과 같은 순서로 진행된다.

1. 사용자가 사용자 인터페이스를 통해서 필요한 관리 정보를 입력한다.
2. 입력된 정보를 바탕으로 에이전트 선택자가 이전의 캐쉬에 있는 에이전트와 비교하여 있을 경우에는 캐쉬의 에이전트를 사용한다. 없을 경우에는 데이타 관리자가 에이전트를 새로 생성한다.
3. 에이전트 이송자가 순회 스케줄 정책에 따라서 이동할 네트워크 컴포넌트의 주소로 이동 에이전트를 보낸다.
4. 에이전트가 네트워크 컴포넌트에 오면 이동 코드 데몬이 그림 6과 같이 이동 에이전트를 받아서 각 해당되는 이동 코드 관리자의 컨텍스트로 보낸다.

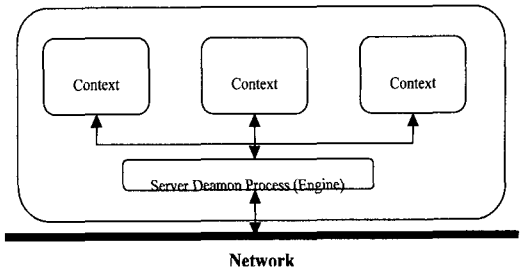

그림 6 데몬과 컨텍스트

5. 이동 코드 관리자는 컨텍스트에서 에이전트 Proxy를 생성한다.
   Proxy와 에이전트의 관계는 그림 7과 같다.

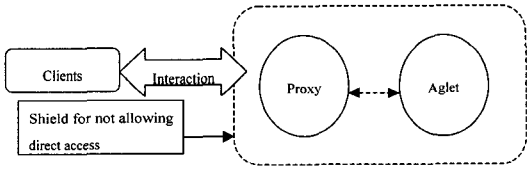

그림 7 에이전트와 Proxy

6. 이동 코드 관리자는 에이전트 Proxy를 통하여 에이전트에서 SNMP Instructions + MIB variables + MIB variable value에 대한 정보, 순회 스케줄 정책, 네트워크 컴포넌트 주소 리스트를 얻어내고, 에이전트를 실행시켜서 관리 정보를 에이전트 조정자에게 보낸다.
7. 에이전트 조정자는 받은 관리정보를 바탕으로 게이트웨이를 통해서 해당 답신을 받아서 이동 코드 관리자에게 보내고, 이동 코드 관리자의 이동코드가 그 정보를 받아서 SNMP Instructions + MIB variables + MIB variable value의 정보를 업데이트한다.
8. 업데이트가 모두 끝나면, 이동 코드 관리자는 에이전트 이송자에 네트워크 컴포넌트 주소 리스트 중에서 아직 방문하지 않은 리스트를 보낸다.
9. 에이전트 이송자는 에이전트의 다음 순회지를 결정하여 에이전트를 Dispatch한다. 기본 순회 정책일 경우에는 다음 순회지를 찾을 수 없으면 Exception 메시지를 발생한다. 만약 순회 정책이 다중 순회 정책으로 되어 있을 경우에는 에이전트의 Clone을 생성하여 보내고, 그에 대한 답신이 오면

원래 에이전트를 제거한다. 답신이 오지 않을 경우에는 다시 Clone을 생성하여 대안(Alternative) 네트워크 컴포넌트로 보내어 신뢰성 있는 전송을 보장한다. 아래의 그림 8은 신뢰성 있는 에이전트 이동 모델을 보여준다.

10. 4에서 9까지의 과정을 거치다가 모든 네트워크 컴포넌트 리스트에 대한 순회가 끝나면 그 끝난 네트워크 컴포넌트에서 에이전트를, 보낸 네트워크 관리 컴포넌트로 보낸다.
11. 네트워크 컴포넌트에 도착하면 데이타 관리자는에이전트에서 필요한 관리 정보를 추출하여 사용자에게 보여주고, 사용자는 정보를 확인하고, 저장하거나 정보를 삭제한다.

에이전트 철회의 경우에는 앞의 순서와 같은 순서로 에이전트가 순회하다가 사용자가 철회 메시지를 각 네트워크 컴포넌트에 보내면, 그 중에서 에이전트가 작동중인 네트워크 컴포넌트의 이동 코드 관리자가 에이전트의 작동을 멈추고, 그 때까지의 작동 결과를 가지고 네트워크 관리 컴포넌트에 보낸다.

그리고, 각 이동 코드 관리자들은 이동 에이전트에 대한 로그 관리를 통해 에이전트의 이동에 대한 정보를 관리한다.

3.3 구현 및 기존 대표 에이전트(Delegated Agent) 방식과의 비교

3.3.1 구현 결과

구현 모델은 JDK1.1.6을 기본 Java 패키지로 하고, Aglet1.0.3 프레임워크를 사용하여 구현하였다. 구현된 모델은 Windows NT Server 4, Windows 95, Windows 98, Linux 등 여러 플랫폼에서 사용할 수 있

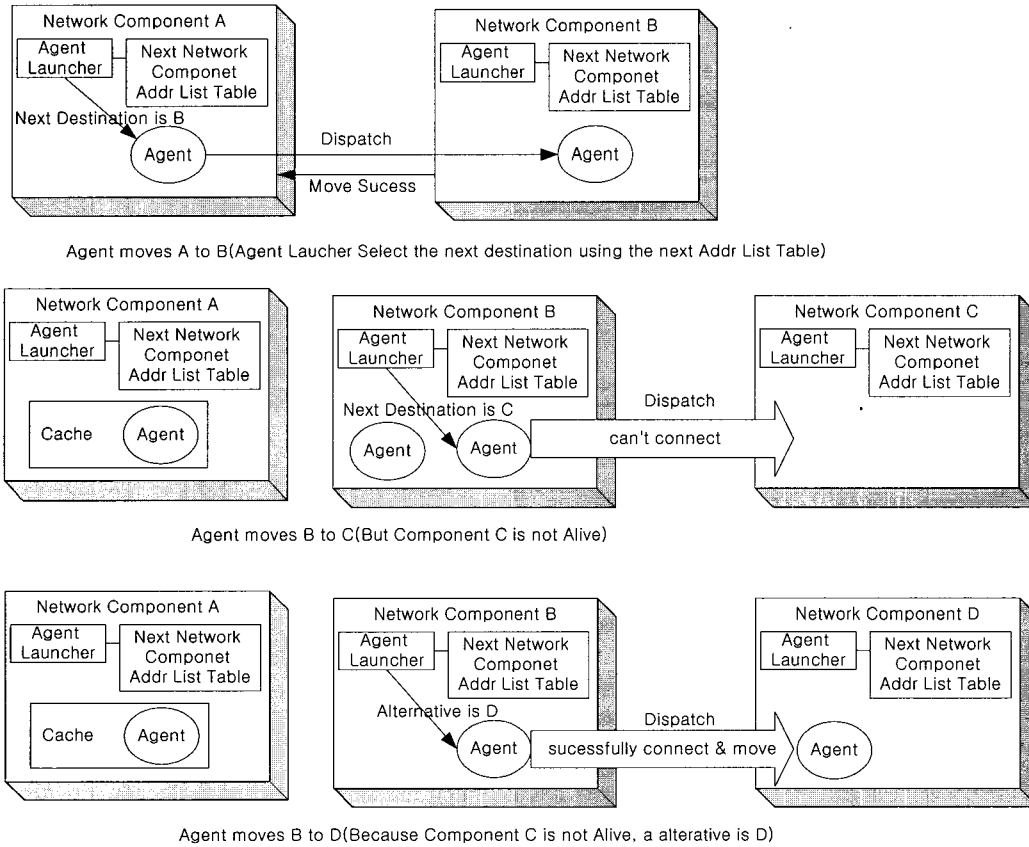

그림 8 신뢰성 있는 에이전트 이동 모델

다. 에이전트는 그림 9의 에이전트 생성 인터페이스를 통하여 생성하며, 생성된 에이전트는 그림 10과 같은 형태로 생성이 된다. 그림 10은 생성된 에이전트들의 리스트를 표현해주고, 그 에이전트들에 대한 관리를 위한 인터페이스이다. 생성된 에이전트들은 그림 10의 Dispatch를 통해서 다른 에이전트 서버로 이동되고, 이동 후에는 에이전트에 입력되어 있는 순회 서버 리스트를 따라서 이동하고 실행된다. 이동 에이전트 코드의 생성은 현재 Aglet 프레임워크와 JDK를 이용해서 미리 구현된 Class 파일을 이용해서 이루어지며, 이 Class 파일은 관리 정보와 관리 서버 정보 등의 정보가 통합되어 Code화된다.

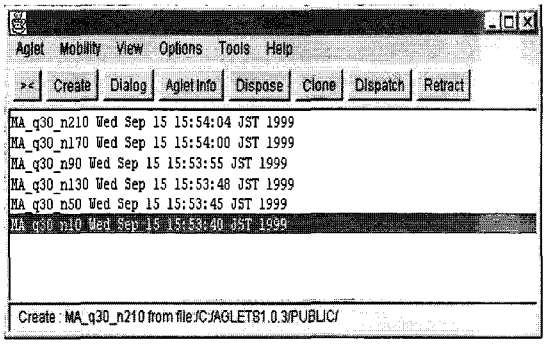

그림 9 에이전트 관리자

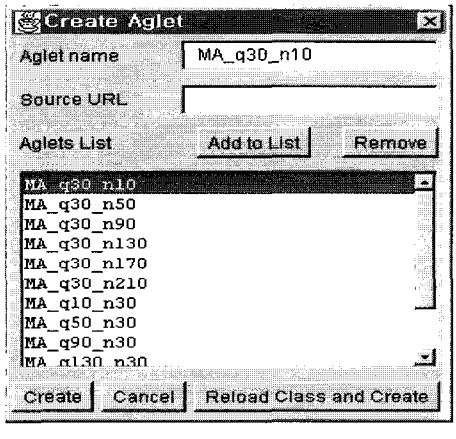

그림 10 에이전트 생성 인터페이스

에이전트의 정보는 그림 11과 같은 형태로 제공된다. 그림 12는 에이전트를 Dispatch하는 인터페이스로, 에이전트 서버 리스트를 제공하고 이 리스트에서 주소를 선택하여 Dispatch를 실행하면 에이전트가 그 서버로 이동한다. 모든 실행이 끝난 에이전트는 그림 9의 에이전트 리스트에 나타나서 순회가 끝났음을 알린다. 관리 정보는 파일 형태로 저장되어 사용자가 결과를 볼 수 있다.

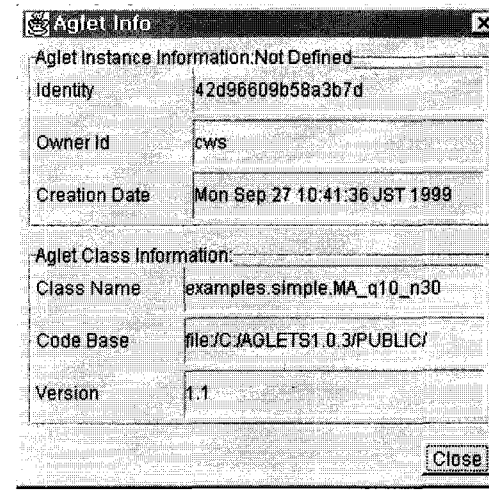

그림 11 에이전트 정보

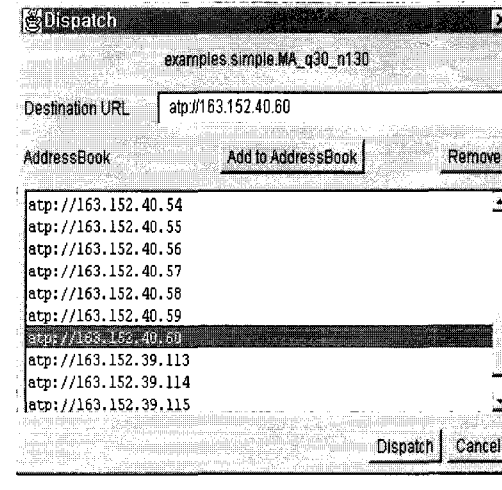

그림 12 에이전트 Dispatch

3.3.2 기존 대표 에이전트(Delegated Agent) 방식과의 비교

기존 대표 에이전트(Delegated Agent) 방식과의 비교 항목은 제공하는 서비스, 에이전트의 기능, 구현 언어, 시스템 구조, 관리 정보 표현 방법, 에이전트 순회 방법 등으로 나누어 비교하였다. 비교 대상은 Webbin[10],

MAGENTA[11], IMA[12]이다. 다음 표 3은 각 모델의 비교결과이다.

표 3 각 모델간 비교

비교항목 \ 비교대상	제안 모델	Webbin	MAGENTA	IMA
에이전트 기능	통합된 관리 정보의 관리	관리 프로토콜의 매핑	통합된 관리 정보의 관리	통합된 관리 정보의 관리
이동 에이전트 순회 방법	Itinerary Policy	없음	Simple Itinerary	Simple Itinerary
시스템 구조	이동 에이전트 + 에이전트 관리자	고정 에이전트	이동 에이전트 + 에이전트 관리자	이동 에이전트 + 에이전트 관리자
구현	Java + Aglet	Java + C	Java	Java
사용자와 에이전트간의 인터페이스의 제공	Interactive한 인터페이스를 제공	Interactive한 인터페이스를 제공	없음	간단한 인터페이스제공
네트워크 fault tolerancy	신뢰성 모델을 제공	없음	있음	알 수 없음
이동 프로토콜	ATP	HTTP	TCP	TCP

4. 각 패러다임의 비교 및 사례 연구

4.1 각 패러다임의 패러메터

각 패러다임의 통신량을 수치화하기 위해서 기본적으로 관리하고자 하는 네트워크에 있는 네트워크 장비의 개수를 N이라고 하고, 그 N개의 장비에 보내지는 질문의 수를 Q개라고 한다. 이럴 때 어플리케이션에서 보내지는 데이타의 크기를 X라고 가정하고, 이 데이타가 네트워크 패킷으로 바뀔 때의 전체 데이타의 크기를 X′라고 하면 $X' = \alpha(X) + \beta(X)X$로 볼 수 있다. 여기서 $\alpha(X)$는 통신 컨트롤을 위한 인포메이션의 크기로 보고, $\beta(X)X$는 데이타 X를 네트워크 패킷으로 바꾼 데이타의 크기로 본다. 이럴 때 $X' = \left\{\frac{\alpha(X)}{X} + \beta(X)\right\}X$ 로 표현될 수 있다. 이 때 $\left\{\frac{\alpha(X)}{X} + \beta(X)\right\}$를 통신 오버헤드 함수로 보고, 이것을 $\eta(X)$로 놓으면, 전체 통신 데이타의 크기인 $X' = \eta(X)X$ 로 표현된다[13]. 단, $\eta(X)$는 항상 1보다 크다.

위의 모델의 기본으로 하여 각 패러다임에 해당되는 통신량의 계산은 다음 아래와 같다.

1) 중앙 집중형 패러다임 모델

중앙 집중형 모델은 다음 식(1)의 형태로 통신량이 표현될 수 있다.

n: 네트워크 장치의 수
q: 질문의 수
I_q : 관리자에서 에이전트로 보내지는 인스트럭션의 크기
R_{qn}: 각 에이전트에서 돌아오는 결과 데이타의 크기
$\eta(I_q)_{CS}$: I_q에 대한 중앙 집중형 모델 통신 오버헤드 함수
$\eta(R_{qn})_{CS}$: R_{qn}에 대한 중앙 집중형 모델 통신 오버헤드 함수

$$T_{cs} = \sum_{n=1}^{N}\sum_{q=1}^{Q}(\eta(I_q)_{CS}I_q + \eta(R_{qn})_{CS}R_{qn}) \qquad (1)$$

즉, 중앙 집중형 패러다임 모델의 통신량은 각 네트워크 장치의 수와 보내지는 인스트럭션의 수만큼의 데이타가 발생하고 그 인스트럭션의 답신에 해당되는 데이타의 합으로 그 총량을 정의할 수 있다.

2) 이동 코드 패러다임 모델

이동 코드 패러다임 중에서 COD(Code On Demand) 실제로 설계된 모델이 찾지 못했기 때문에 비교 대상에서 제외하였다. 그리고, 이동 코드 패러다임 모델에서도 위의 중앙 집중형 모델과 마찬가지로 $\eta(X)$는 통신 오버헤드 함수를 의미하고, R은 하나의 인스트럭션에 해당되는 답신의 크기이다. C는 이동 코드의 크기를 의미한다.

(1) REV(Remote EValuation)

REV의 경우에는 네트워크 장치로 보내지는 코드에 모든 질문에 대한 것이 합쳐져서 보내지므로, 중앙 집중형 모델과 달리 답신에 드는 코드의 비용이 하나의 통합된 질문으로 처리된다. 그러므로, 전체적인 통신량은 식 (2)와 같이 정의된다.

$$T_{REV} = \sum_{n=1}^{N}(\eta(C_{REV})_{REV}C_{REV} + \eta(R_{qn})_{REV}\sum_{q=1}^{Q}R_{qn}) \qquad (2)$$

n: 네트워크 장치의 수
q: SNMP 질문의 수
C_{REV} : 관리자에서 에이전트로 보내지는 통합된 인스트럭션의 크기
$\sum_{q=1}^{Q}R_{qn}$: 각 에이전트에서 돌아오는 통합된 결과 데이타의 크기
$\eta(C_{REV})_{REV}$: C_{REV}에 대한 REV 모델 통신 오버헤드 함수
$\eta(R_{qn})_{REV}$: $\sum_{q=1}^{Q}R_{qn}$에 대한 REV 모델 통신 오버헤드 함수

(2) MA(Mobile Agent)

MA의 경우에는 MA가 자율적으로 네트워크 에이전트 서버를 돌아다니므로, MA가 방문해야 할 네트워크

장비의 개수에 따라 생성되어 다음 네트워크 에이전트 서버로 보내지는 코드의 크기가 변한다. 이동 코드의 크기는 초기에 보내지는 이동 코드인 C_{MA}와 이동 코드가 에이전트 서버에서 작동 결과인 D_{MA}의 합이 된다. 그런데, D_{MA}는 각 서버에서 실행될 때마다 그 크기가 증가하므로 서버의 개수에 따라 다음 아래의 식과 같이 정의될 수 있다.

$$D_{MA,n} = \begin{cases} 0 \quad (\text{if } n = 1) \\ \sum_{m=1}^{n-1} \sum_{q=1}^{Q} R_{qm} \quad (\text{if } n > 1,\ m\text{은 에이전트 서버의 개수}) \end{cases} \tag{3}$$

위의 식을 바탕으로 이동 에이전트의 이동시의 전체 통신량은 아래 식으로 표현된다. 식에서 네트워크 에이전트 서버의 총 개수를 N으로 보았을 때 실제 발생하는 에이전트의 이동은 N+1번 이루어지므로, 전체 통신량의 계산에 있어서 N+1로 정의된다.

n: 네트워크 장치의 수
q: SNMP 질문의 수
C_{MA}: 관리자에서 에이전트 서버로 보내지는 초기 이동 코드 크기
D_{MA}: 에이전트 서버에서 작동 결과의 크기
$\eta(C_{MA}+D_{MA,n})_{MA}$: $C_{MA}+D_{MA,n}$에 대한 MA 모델 통신 오버헤드 함수

$$T_{MA} = \sum_{n=1}^{N+1} \eta(C_{MA}+D_{MA,n})_{MA}(C_{MA}+D_{MA,n}) \tag{4}$$

3) 분산 객체 패러다임

분산 객체 패러다임의 경우에 있어서는 그 목적이 여러 다른 시스템의 통합에 그 목적이 있기 때문에, 그 사용할 때의 구조를 보면, JMAPI, CORBA, WBEM, ODMA의 경우에 모두다 다른 패러다임을 사용하고 있다. JMAPI의 경우에는 RMI를 이용한 REV 패러다임을 사용하고 있으며, WBEM의 경우에는 트랜스포트 프로토콜만을 HTTP를 사용할 뿐 그 형태는 중앙 집중형 패러다임을 사용한다. CORBA, ODMA의 경우에는 하나의 패러다임이 아닌 여러 분산 패러다임을 적용한 형태가 존재한다. 그러므로, 그 일반화된 통신량의 계산이 어렵고, 기본적으로 분산 객체 패러다임 외에 사용된 패러다임 통신량 모델을 사용하는 것이 타당하다고 생각된다.

4.2 각 비교 대상 모델의 전체 통신량 계산

본 논문에서는 본 논문에서 제시하는 분산 에이전트 모델과 REV 모델인 JMAPI, 중앙 집중형 모델인 SNMP 모델의 통신량을 비교한다. 통신량은 각 네트워크 장치의 개수와 질문의 개수에 따른 전체 통신량의 크기를 비교해본다. 4.1 장에서 보았던 식(1), (2), (4)를 이용해 그 변화량을 알아본다. 그리고, 사용된 SNMP 모델은 AdventNet사의 SNMP 패키지를 사용하였다.

CS 모델: $T_{CS} = \sum_{n=1}^{N} \sum_{q=1}^{Q} (\eta(I_q)_{CS} I_q + \eta(R_{qn})_{CS} R_{qn})$ (1)

REV 모델: $T_{REV} = \sum_{n=1}^{N} (\eta(C_{REV})_{REV} C_{REV} + \eta(R_{qn})_{REV} \sum_{q=1}^{Q} R_{qn})$ (2)

MA 모델: $T_{MA} = \sum_{n=1}^{N+1} \eta(C_{MA}+D_{MA,n})_{MA}(C_{MA}+D_{MA,n})$ (3)

각 모델의 오버헤드 패러메터는 예를 들어 q=30, n=10이고, MIB Integer 변수 10개에 대한 get과 get response에 대해서 계산을 하면, (1)식의 $\eta(I_q)_{CS}$는 생성되는 SNMP 메시지의 크기는 102 byte 이고 이에 대한 UDP와 IP 헤더에 대한 오버헤드는 28 byte 이므로 오버헤드 함수 $\eta(I_q)_{CS}$는 $\frac{IP\ datagram\ size}{SNMP\ message\ size} = \frac{130}{102} = 1.27$, $\eta(R_{qn})_{CS}$도 마찬가지 방법에 의해 1.27이된다.

(2)식의 $\eta(C_{REV})_{REV}$는 생성되는 HTTP 메시지의 크기는 178 byte이고 이에 대한 TCP와 IP 헤더의 오버헤드는 40 byte 이므로 오버헤드 함수 $\eta(C_{REV})_{REV}$는 $\frac{IP\ datagram\ size}{HTTP\ message\ size} = \frac{218}{178} = 1.22$ 가 되고, $\eta(R_{qn})_{REV}$도 마찬가지 방법으로 구하면 1.27이 된다.

(3)식의 $\eta(C_{MA}+S_{MA,n})_{MA}$는 생성되는 관리 에이전트 클래스의 크기가 3366byte 이고 이에 대한 ATP, TCP와 IP 오버헤드는 MTU= 1500으로 가정했을 때, 4개의 IP datagram으로 세그멘테이션(segmentation)되므로, (40 byte + 16 byte)* 4 = 224 이고, $\eta(C_{MA}+S_{MA,n})_{MA}$ 는 $\frac{IP\ datagram\ size}{Agent\ class\ size} = \frac{3590}{3366} = 1.07$ 이 된다.

4.3 사례 연구

4.2 의 모델을 기준으로 사용된 관리 정보는 RFC 1213 MIB를 사용하였고 (IfIndex, SysService, ifMtu, ifNumber, ifSpeed, ifOperStatus, ifInUcastPkts, ifNUcastPkts, ifInDiscards, ifInErrors, ifOut-UcastPkts, ifOutNUcastPkts, ifOutDiscards, ifOut-Errors) 14개의 Integer 변수 항목에 대해서 일반적인 값 요구와 그에 대한 답신에 대한 통신량을 실험하여 계산하였다. MA 모델에서의 SNMP로의 매핑은 정적으

로 미리 정의해서 사용하였다. 그림 13은 네트워크 장치 수에 따라 나타나는 통신량을 실험한 것으로, 각 장치에 보내지는 SNMP query의 횟수인 Q가 30에 해당되는 관리 정보 요구에 대해서, 네트워크 장치 수인 N을 점차 증가시켜 나타난 그래프이다. 그래프의 단위는 Q의 경우에는 SNMP의 경우를 기준으로 하여 표현하였다. 그러므로 MA와 REV의 경우는 SNMP Q=30인 크기만큼의 통합된 요구와 답신이 이루어진다.

그림 13의 그래프를 살펴보면, REV 모델은 네트워크 장치의 수만 증가할 때, SNMP와 비슷한 정도의 통신량을 생성하고, MA 모델의 경우에는 SNMP나 REV 모델보다 낮은 통신량 증가율을 보인다. 그래프에서 Y 축의 단위는 Kbyte이다.

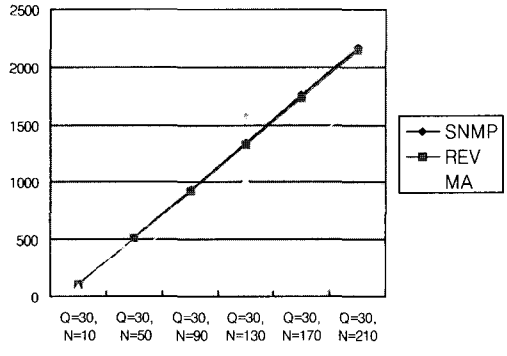

그림 13 네트워크 장치의 수에 따른 통신량

그러므로 MA 모델은 그 크기에 상관없는 특성에서 SNMP나 REV 모델보다 좋은 성능을 보임을 알 수 있다.

다음 장의 그림 14의 그래프는 질문의 수에 따라나타나는 통신량을 계산한 것으로, N = 30 으로 고정시키고, Q를 점차 증가시켜 나타난 그래프이다.

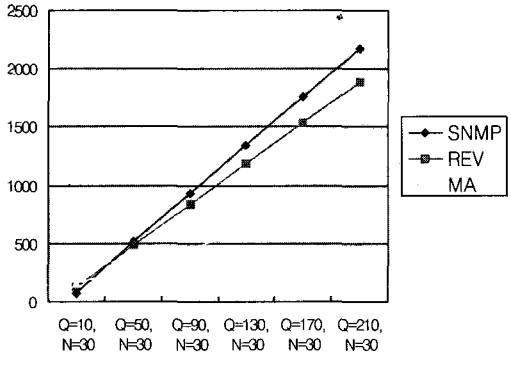

그림 14 질문의 수에 따른 통신량

그림 14를 살펴보면, REV 모델과 MA 모델 모두 SNMP 모델에 비해 그 통신량이 적게 증가하는 것을 알 수 있다. 두 모델 모두 MbD 모델에 기반했기 때문에 관리자와 에이전트간의 통신이 줄어든 결과로 볼 수 있다. 그리고, REV 모델에 비해 MA 모델이 현저하게 통신 증가율이 작은 것을 알 수 있는 데. 이것은 MA 모델의 질문에 의한 이동 코드의 증가가 REV 모델에 비해 적고, 질문에 대한 답신이 마지막에 한 번만 일어나기 때문이다.

두 개의 그래프에서 MA 모델은 네트워크 크기에 제한 받지 않고, MbD 모델에 충실하게 작동함을 알 수 있다. 표 4는 각 모델을 비교한 것이다. 표에서 Scalability는 SNMP의 $\frac{N\text{의 변화량}}{\text{통신량의 변화량}}$, $\frac{Q\text{의 변화량}}{\text{통신량의 변화량}}$ 으로 된 기울기를 1로 보았을 때의 상대적 기울기로 표시하였으며, 결과를 보면 MA 모델이 앞의 두 모델보다 확장성(Scalability)이 좋은 것을 알 수 있다.

표 4 각 모델 비교

	Network device Scalability	Management Instruction Scalability	Transport Protocol	Mobile code Type
SNMP	1	1	UDP	No mobile
REV	1.02	1.25	TCP	RMI
MA	1.27	3.15	TCP	ATP

5. 결 론

본 논문에서는 네트워크 관리에 관한 패러다임을 알아보고, 특히 분산 패러다임의 적용에 대해서 살펴보았다. 그리고, 이동 에이전트 패러다임을 이용한 네트워크 관리 모델을 제안하였다. 제안한 네트워크 관리 모델은 자바에 기반한 이동 에이전트 프레임워크인 Aglet을 이용해서 설계되어 상호운용성을 제공하고, MbD 모델에 기초하여 설계되었기 때문에 크기에 제한 받지 않으며, 유연성과 확고성을 제공한다. 또한 Aglet은 MAF(Mobile Agents Facility)에 기초한 분산 오브젝트 프레임워크이므로 CMIP와 SNMP와 같은 기존의 시스템과의 운용에 필요한 인터페이스를 정의하여 기존 시스템과의 연결이 가능하고, 관리 오퍼레이션이 소프트웨어로 처리될 수 있으므로 소프트웨어 변환 통해 여러 다른 형태로의 변환이 가능하다. 또한, 분산 패러다임의 네트워크 관리에의 적용의 실효성을 알아보기 위해서

각 분산 패러다임을 패러메터화된 통신량 모델을 이용해서 비교해 보았다. 적용한 패러메터화된 통신량 모델은 각 네트워크 장비와 질문의 수, 관리자와 네트워크 장치간의 교환되는 메시지와 코드의 양을 통한 모델이다. 실험 비교 결과 제안한 MA 모델은 네트워크 장치의 수와 질문의 수에 크게 통신량이 증가하지 않는 크기에 제한 받지 않는 특성을 보여주었고, 또한, 관리자와 네트워크 장치간의 통신량이 줄어드는 MbD 모델의 특성에 일치하고 있음을 알 수 있었다. 결과적으로 분산되어 있는 대규모의 네트워크의 관리에는 MA 패러다임의 적용이 타당하다고 할 수 있다. 또한 제안하는 모델은 에이전트의 신뢰성 있는 전송을 보장한다.

앞으로의 연구에서는 현재 제안한 MA 모델에서 SNMP와 CMIP에 대한 매핑을 동적으로 처리하는 부분에 대한 보완, 통신 효율성의 비교를 위한 질문에 대한 답신에 대한 시간 측정, 이동 에이전트의 효율적인 전송과 실행에 대한 연구가 더 필요하다.

참 고 문 헌

[1] 유승근, 정진욱, "웹 기반 관리 표준 기술 분석", 개방시스템, 개방형컴퓨터통신 연구회, 제12권, 제4호,pp.6-12, 1998

[2] G. Goldszmidt and Y. Yemini., "Distributed Management by Delegation," *In Proc. of the 15th Int. Conf. on Distributed Computing*, June, 1995

[3] Danny B. Lange and Mitsuru Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, p17-134, Addison-Wisley, 1998

[4] J.-P. Marthin-Flatin and S. Znaty., "A Simple Typology of Distributed Network Management Paradigms," In *Proc. of the 8th IFIP/IEEE Int. Workshop on Distributed Systems: Operations & Management (DSOM '97)*, Oct, 1997

[5] J. Case, M. Fedor, M. Schoffstall and J. Davin(Eds.)., RFC 1157. "A Simple Network Management Protocol(SNMP)," IETF, May, 1990

[6] Warrier, L. Besaw, RFC 1095. "Common Management Information Services and Protocol over TCP/IP (CMOT)," U.S. Apr, 1989

[7] S. Waldbusser, RFC 1757. "Remote Network Monitoring Management Information Base," February, 1995

[8] F. Fuggetta, G.P. Picco and G. Vigna., "Understanding Code Mobility," IEEE Trans. on Software Engineering. Berlin, Germany, 1997

[9] 최원상, 김태윤, "이동 에이전트 패러다임에 기반한 망 관리 서비스의 설계," 한국정보처리학회 춘계학술발표회, 제6권, 제1호, pp.1099-1102, 1999.

[10] Luca Deri "Webbin': A New Way To Manage Networks," Linux Magazin, September 1997

[11] Sahai, A., Morin C. "Mobile Agents for Managing Networks : MAGENTA perspective," Chapter 15, Software Agents for Future Communications Systems. A.L.G.Hayzelden and J. Bigham (Eds), Springer Verlag. ISBN 3-540-65578-6. April 1999

[12] Dr. Luderer, Hosoon Ku, Baranitharan Anand, "Network Management Agents Supported by a Java Environment," ISINM'97, SanDiego CA. 1997

[13] G.P. Picco and Mario Baldi, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications," In Proceedings of the 20th International Conference on Software Engineering (ICSE'98), Kyoto (Japan), R. Kemmerer and K. Futatsugi, eds., IEEE CS Press, ISBN 0-8186-8368-6, pp. 146-155., April, 1998

최 원 상

1998년 고려대학교 컴퓨터학과 학사. 1998년 ~ 고려대학교 컴퓨터학과 대학원 석사과정. 관심분야는 컴퓨터 네트워크, 분산 컴퓨팅, 네트워크 관리, 이동 컴퓨팅

김 태 윤

1981년 고려대학교 산업공학과 학사. 1983년 Wayne state University 전산과학과 석사. 1987년 Auburn University 전산과학과 박사. 1998년 ~ 현재 고려대학교 컴퓨터학과 교수. 관심분야는 전자상거래, 컴퓨터 네트워크, EDI, 이동통신 등