논문 02-27-6C-1 한국통신학회논문지 '02-6 Vol.27 No.6C

분산처리환경에서 이동에이전트를 이용한 효율적인 망 관리 구조

학생회원 이 정 환*, 정회원 홍 충 선*

An Efficient Network Management Architecture Using Mobile Agents on DPE

Jung Hwan Lee** *Student Member*, Choong Seon Hong** *Regular Member*

요 약

기존의 단순한 객체를 이용한 중앙집중형 망관리 시스템이 가지는 단점들을 극복하고자 분산객체를 이용하는 동적 객체 플랫폼이 대안으로 제안되었고 다양한 분산객체 기술, 즉 CORBA나 Java-RMI를 이용한 분산 망 관리 시스템들이 구현되었다. 그 후 CORBA 기반이나 Java-RMI 기반의 시스템들이 가지지 못했던 시스템의 확장성과 유연성을 제공하기 위해 이동 에이전트 기반의 플랫폼이 제시되었으나 이동 에이전트의 사용으로 추가적으로 발생할 수 있는 트래픽의 문제, 망 요소의 리소스 관리 문제 등을 해결하고 고려해야 한다. 따라서, 본 논문에서는 이동 에이전트를 이용한 효율적인 망 자원을 구조를 제안하였다. 제안한 구조에서는 이동 에이전트가 이동할 때 발생할 수 있는 트래픽의 최소화 방안, 망 요소의 효율적인 리소스 관리 및 이동에이전트의 성능 향상을 위해서 TMN의 정보구조를 사용한다.

ABSTRACT

In order to overcome disadvantages of existing centralized network management systems that use simple objects, dynamic object platforms are proposed as alternative systems. So the distributed network management systems are implemented using various distributed platform such as CORBA and Java-RMI. Subsequently mobile agent-based platforms are proposed. The mobile agent-based platform can additionally provide flexibility and scalability to network management system that CORBA or Java-RMI based platforms do not support. In this paper, we address an architecture to solve the problem of the occurrence of additional traffic by using mobile agent and to save resources of network element.

Our proposal in this paper is an efficient network management architecture using mobile agents. And we make use of mobile agents for minimizing of traffic that can happen when mobile agents moves. Also we design agents using information architecture of TMN for efficient resource management of network element and improvement of operation performance.

I. 서 론

인터넷의 대중화와 함께 네트워크 트래픽은 계속적으로 증가하고 있다. 이러한 과도한 트래픽의 영향으로 망 자체가 마비되거나 지나친 응답 시간의 저하가 초래될 수 있다. 이러한 문제점을 해결하기 위해 망을 지속적으로 관리하고 모니터링 및 분석을 해주는 것은 이제 필수적인 작업이 되었다. ISO (International Organization for Standardization)에서는 CMIS / CMIP (Common Management Informa-

* 경희대학교 전자정보학부(nowsys@networking.kyunghee.ac.kr, cshong@khu.ac.kr)
논문번호 : 010394-1214, 접수일자 : 2001년 12월 14일
※ 본 연구는 한국과학재단 목적기초연구(R05-2001-000-00976-0) 지원으로 수행되었음.

tion Protocol)[1]을 IETF (Internet Engineering Task Force) 에서는 SNMP (Simple Network Management Protocol)[2]을 각각 제안하였다. 이들은 전형적인 클라이언트 / 서버 형태의 중앙집중화된 구조이다. 관리자와 에이전트로 구분되어지는 이 구조는 비교적 구현이 쉽고 단순한 구조를 가지고 있어 많은 망 요소 제공자(Network Element Vendor)들이 이러한 구현 모듈을 탑재하여 보급하고 있다. 하지만 이런 중앙 집중화 된 구조의 문제는 WAN (Wide Area Network)과 같은 광범위한 네트워크 상에서 관리할 때 병목현상 (Bottleneck)이 발생하여 여러 개의 서브 네트워크를 실시간으로 관리하고 모니터링하기 어렵다는 문제점을 가지고 있다.[3] 이러한 중앙 집중화 된 관리구조의 문제를 해결하기 위해서 MAS (Multi-Agent System)[4]를 지원하는 분산처리환경(Distributed Processing Environment)의 구조가 제안되었다.

여러 개의 분산된 객체들이 각각의 에이전트에 고정되어 있는 형태가 바로 CORBA[5] 혹은 Java-RMI[6]를 이용한 고정 객체 플랫폼이며, 이와는 상반되는 것이 바로 이동 에이전트를 이용한 동적 객체 플랫폼이라 할 수 있겠다. 물론 객체의 생성과 소멸로만 생각한다면 CORBA나 Java-RMI 역시 동적 객체에 속하지만 여기서 언급하는 '동적'이란 자유로운 객체의 생성과 소멸 그리고 객체의 이동성 보장까지 포함한다. 이동 에이전트란 물리적인 네트워크의 위치를 동적으로 이동할 수 있는 소프트웨어 컴포넌트를 일컫는다. 이러한 동적 객체 플랫폼이 정적 객체 플랫폼에 비해서 가질 수 있는 장점은 유연성과 확장성이라고 할 수 있을 것이다. 이 논문에서는 동적 객체 플랫폼인 이동 에이전트 플랫폼이 가지는 장점들을 활용하고 이동 에이전트가 가질 수 있는 단점, 즉 이동 에이전트가 이동할 때 발생하는 추가적인 트래픽의 발생문제 그리고 Network Element의 효율적인 리소스 관리에 대해서 연구하였다. 논문의 구성은 다음과 같다. 제 2장에서는 망관리를 위해 제안되었던 이동 에이전트를 이용한 관련 연구에 대해서 알아보고, 제 3장에서는 이동 에이전트와 고정 에이전트의 설계와 구현에 대해서 기술하며, 제 4장에서는 설계된 이동 에이전트와 고정 에이전트를 바탕으로 한 통합 이동 에이전트 플랫폼의 설계와 구현 대해서 기술하며 5장에서는 제안한 사항에 대한 평가를 기술하겠으며 마지막 6장에서는 결론과 앞으로의 연구 과제에 대해서 설명한다.

Ⅱ. 이동 에이전트를 이용한 망관리 플랫폼

이번 장에서는 이동 에이전트를 이용해서 망관리에 제안되었던 기존의 연구에 대해서 설명하도록 하겠다. 이동 에이전트를 망관리에 사용하면 결함(Fault) 관리, 계정(Account) 관리, 구성(Configuration) 관리, 성능(Performance) 관리, 보안(Security) 관리의 망관리 기능적 분류 요소들을 사용할 때 에이전트를 각 요소마다 배치하여 불필요한 동작을 줄일 수 있으며 이종 환경을 지원하는 등의 장점을 가질 수 있다.

표 1은 이동 에이전트를 이용한 망관리에서 가질 수 있는 장단점에 대해서 정리하였다.

표 1. 이동 에이전트 기반의 망관리 시스템의 장단점

항 목	내 용
장 점	① 효율성 극대화 (각 요소별 에이전트 배치) ② 이동 에이전트의 이동에 따른 Network Element의 저장공간 절약 ③ 이종환경에 대한 지원 ④ 확장성 ⑤ 쉬운 소프트웨어 업그레이드
단 점	① 이동 에이전트의 이동 시 추가적인 트래픽 발생 ② 이동 에이전트의 이동 가능한 영역 지정문제

2.1 망관리에 적용된 플랫폼 구조

망관리에 적용된 이동 에이전트 플랫폼의 구조는 일반적으로 그림 1과 같다. 망관리자에 의해 생성된 이동 에이전트는 이동 시나리오[7]에 의해서 이동하며 주어진 작업을 수행하게 된다. 이동 에이전트의 수행 동작을 정의하는 시나리오는 크게 다음과 같은 2가지로 분류할 수 있다.

① GnG(Get and Go) : 관리자에 의해 생성된 이동 에이전트는 목적지 노드로 이동 후 주어진 작업을 수행한다. 작업이 끝나면 관리자에게 곧바로 다시 돌아오는 것이 아니라 다음 목적지 노드를 찾기 위해 다시 이동하게 된다. 각 해당 노드마다 같은 작업을 수행할 수도 있고 관리자의 세부 시나리오에 따라 다른 작업을 수행 할 수도 있다.

② GnB(Get and Back) : 관리자에 의해 생성된 이동 에이전트는 목적지 노드로 이동한다. 이동 후 주어진 작업을 수행 후 바로 관리자에게 돌아온다.

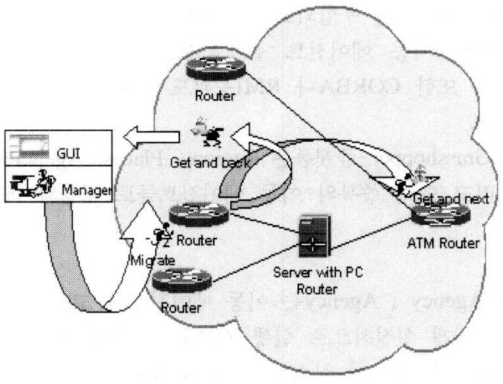


그림 1. 망관리에 이동 에이전트를 적용한 일반적인 구조

위와 같은 단순 동작 시나리오를 가지는 구조로는 이동 에이전트의 관리, 망의 성능관리, 데이터들의 효율적 관리에 문제점이 있기 때문에 이러한 점을 보완하기 위해 새로운 구조와 알고리즘을 가지는 플랫폼들이 제안되었다.

2.2 MIAMI Project

MIAMI Project [8]는 ACTS[9] (Advanced Communications Technology and Services)의 일부분이며 ACTS란 유럽연합의 발전을 위해서 통합된 기반구조를 만들기 위한 하나의 과제이다. 개방형 유럽 시스템과 통합적 정보 자원구조(Global Information Infrastructure)의 출현은 유럽 연합이 관리해야 할 자원의 복잡성, 정보의 빠른 동적 변화와 자원의 광범위 분포를 초래하게되었다. 그래서 그들은 이러한 환경을 관리하기 위해서 약간의 지능을 가지는 이동성 에이전트 시스템이 필요하게 되었다.

MIAMI(Mobile Intelligent Agents in the Management of the information Infrastructure)는 이러한 복잡한 개방형 유럽 정보 자원의 관리와 망관리, 서비스 관리에서 이동 에이전트의 사용 가능성과 영향을 평가하기 위한 프로젝트이다. 또한 MIAMI 프로젝트의 목적은 다음과 같다.

⊙ 개방형 유럽 정보 구조(Open European Information Infrastructure)의 요구사항을 수렴하고 OMG (Object Management Group)의 MASIF(Mobile Agents Standard Interoperability Facility)에 부합하는 표준화된 약간의 지능을 가지는 이동 에이전트를 개발하는 것

⊙ Open EII(European Information Infrastructure)의 관리 솔루션에 입각한 진일보된 통신과 정보 서비스를 제공할 수 있는 이동 에이전트 시스템을 개발하는 것.

⊙ Pan-European 비즈니스 환경의 서비스 솔루션에 대한 평가와 단일화된 MIA 프래임워크의 참고 구현 모델을 개발하는 것

⊙ 다음 두 가지 권장사항을 제시하는 것

○ 기반사업자와 종단 사업자에게 : 언제 어디서 그리고 어떻게 그들의 향후 제품에 MASIF를 적용시킬 수 있을 것인가.

○ 서비스 제공자에게 : MIA에 입각한 EII 관리 솔루션과 진일보된 통신, 정보 서비스의 제공을 어떻게 개발할 것인가.

⊙ 기존 ACTS 프로젝트의 결과들을 MIA기반 솔루션으로 통합하는 것.

⊙ 논증과 증명, 그리고 출판 등으로 표준화 기구와 포럼에 결과를 배포하고 표준화 작업을 하는 것.

이 연구에서는 앞에서 언급한 망 요소의 효율성 관리를 위해 호스트에 고정 에이전트(Stationary Agents)를 배치시키고 AVP(Active Vitural Pipe)를 통해 망 요소의 동적이고 효율적인 이동 에이전트 플랫폼(그림 2)을 제시하였다[10].

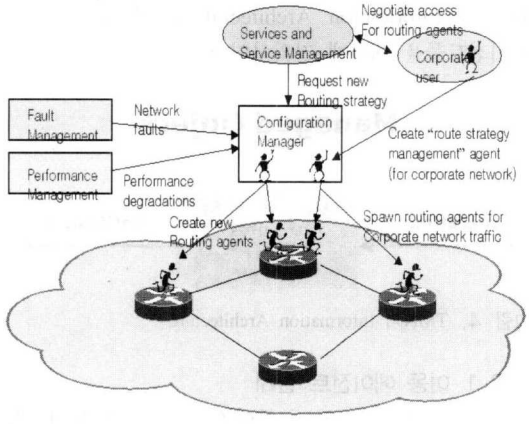


그림 2. MIAMI 플랫폼의 구조

그림 3은 MIAMI의 AVP를 나타낸 것이다. Dynamic Connectivity Manager는 Network Element로 전송되어지는 이동 에이전트의 형태에 따라서 각각의 관리 구성요소들에게 동적인 연결을 제공하게 된다.

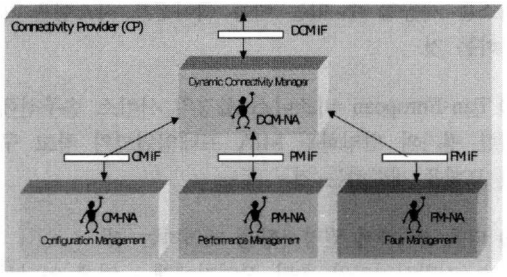


그림 3. MIAMI Active Virtual Pipe Domain

Ⅲ. 제안하는 에이전트별 설계 및 구현

이동 에이전트 플랫폼에서 에이전트는 크게 두 개의 분류로 나누어지는데 첫 번째는 이동 에이전트(Mobile Agents)이고 두 번째는 고정 에이전트(Stationary Agents)이다. 이동 에이전트는 이동성을 가지고 목적지 노드를 방문해 필요한 정보를 수집하는 것이고 고정 에이전트는 이동 에이전트와 반대로 동적으로 다른 지역으로 이동할 수 없는 반면에 특정한 한 지역을 관리하는 에이전트이다. 모든 에이전트는 효율적인 관리를 위해 기본적으로 TMN의 Information Architecture[11]를 사용하였고 TMN의 Information Architecture는 아래 보이는 그림 4와 같으며 이 구조를 OSI의 기능적 관리 영역(Functional Management Area)에 적용하여 장애관리, 성능관리(Performance Management)의 클래스를 TMN의 Information Architecture를 이용하여 고정 에이전트들을 새롭게 정의하였다.

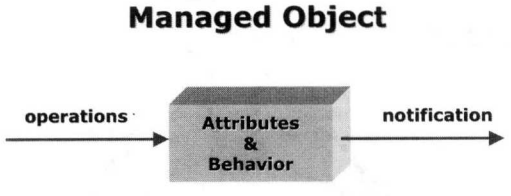


그림 4. TMN의 Information Architecture

3.1 이동 에이전트 설계

이동 에이전트의 플랫폼은 Aglets[12], Grasshopper[13], Voyager[14] 등 각 Vendor마다 특징을 가지는 다양한 플랫폼이 있지만 우리는 MIAMI 프로젝트의 보고서에 추천되어 있는 IKV++사의 Grasshopper 플랫폼을 이동 에이전트 플랫폼으로 사용하였다. Grasshopper는 OMG의 표준에 맞춰서 설계된 이동 에이전트 플랫폼이다. MASIF(Mobile Agents System Interoperability Facility)[15]에 부합되도록 설계되었기 때문에 다른 이종의 이동 에이전트 플랫폼과도 연계되어질 수 있고 또한 CORBA나 RMI등과도 상호 운용할 수 있다.

Grasshopper 플랫폼은 Regions, Places, Agencies 그리고 여러 종류의 이동 에이전트들로 구성되어있다.

⊙ Agency : Agency는 이동 에이전트나 고정 에이전트에게 실질적으로 실행 가능한 환경을 제공하는 요소이다. 때문에 최소한 한 개의 Agency가 호스트에서 동작중이라면 에이전트의 실행을 보장할 수 있다.

⊙ Region : Region의 개념은 Grasshopper 플랫폼 환경에서 분산 컴포넌트들의 관리를 수월하게 한다.

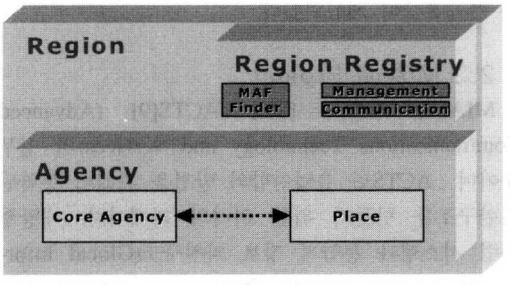


그림 5. Grasshopper 플랫폼의 구조

⊙ Region Registry : Region Registry는 특정한 지역과 연관되어진 모든 컴포넌트들의 정보를 유지하고 있다. 새로운 컴포넌트가 만들어지면 (ex, Agency, place or agent) 그에 해당하는 Registry에 자동적으로 존재하게 된다.

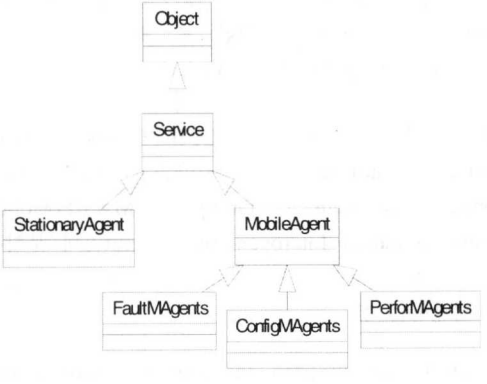


그림 6. Mobile Agents Class의 계층도

기본적으로 이동 에이전트도 동작과 속성을 가지는 객체이다. 따라서 이동 에이전트의 불필요한 동작과 속성을 줄일 수 있다면 이동 에이전트가 이동시 발생하는 추가적인 트래픽을 줄 일 수 있다. 그래서 우리는 망관리 기능들을 5개의 분류 항목으로 나누었고 그 중에서 다시 3개의 분류를 선택하였다. (그림 6) 이렇게 분류된 3개의 항목의 이동 에이전트들은 자신과 연관된 데이터값(표 2)들을 가지는 고정 에이전트와 서로 통신을 한다. 그리고 이동 에이전트는 위에 계층도(그림 6)에서 본 것과 같은 분류에 의해서 생성되며 관리자의 요청에 따라 해당 Network Element로 보내지게 된다.

3.2 고정 에이전트의 설계

고정 에이전트의 구현은 기본적으로 JMX[16] (Java Management Extension) 기반으로 이루어졌다. JMX는 자바 프로그래밍 언어에 있어 망관리와 어플리케이션의 서비스 구조에서 사용되는 API, 디자인 패턴 그리고 구조들을 정의하고 있다. JMX의 구조는 다음과 같이 3가지 레벨로 분류할 수 있다.

- ⊙ 관리 레벨 (Instrumentation level)
- ⊙ 에이전트 레벨 (Agent level)
- ⊙ 분산 서비스 레벨 (Distributed level)

관리 레벨에서는 JMX에서 관리될 자원의 명세를 제공한다. JMX에서 관리될 자원은 어플리케이션이 될 수도 있고 구현된 서비스가 될 수 있고, 또한 특정 장치, 사용자 기타 등등의 것들이 될 수 있다. 주어진 자원의 관리는 하나 혹은 그 이상의 MBeans의 형태로 제공되어진다. 일반적인 MBeans은 JavaBeans 컴포넌트 모델에서부터 파생된 일정한 디자인 패턴을 가지는 자바 객체를 말하는 것이다. 기본적인 JMX의 구조는 그림 7에서 보여주고 있다. MBean의 형태는 TMN의 Information Architecture와 유사하다. 또한 TMN의 Information Architecture는 객체 지향적 접근을 하였고 OSI의 관리 정보 모델에 기반하고 있다.

우리는 MBean 모델을 고정 에이전트의 효율적인 관리를 위해 고정 에이전트의 디자인에 적용시켰다. 그리고 3개의 PerforMSAgents, ConfigMSAgents 그리고 FaultMSAgents의 MBeans 모델을 정의하였다. 그림 9에서는 고정 에이전트의 클래스 계층도를 보여준다.

MIB(Managed Information Base) 값들은 RFC

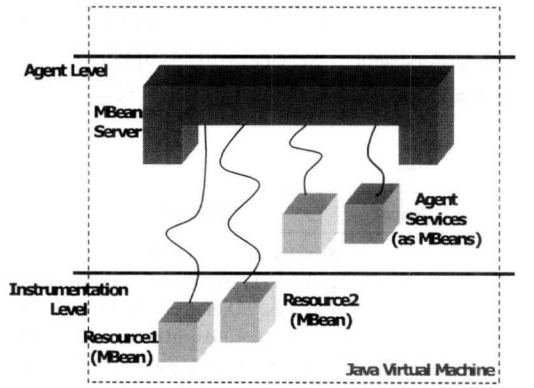


그림 7. JMX의 기본 구조

1213에 정의되어 있다. 우리는 RFC 1213에 정의되어 있는 MIB 값들을 OSI 기능적 분류에 의해 각각의 MBeans의 데이터로 재분류하였다[17]. 재분류는 표 2에서와 같이 하였으며 각각의 MBeans의 특성에 따라 관리할 수 있는 MIB값들을 재분류하였고 MBean은 이 값들을 관리하게 된다. 또한 ManagedStationaryAgent는 PerforMSAgents, ConfigMSAgents, FaultMSAgents를 관리하게 된다. 또한 Network Element의 확장성을 고려하여 JMX에서 제공하는 SNMP API를 사용한다면 SNMP를 지원할 수도 있고 WBEM(Web-Based Enterprise Management)[18] 역시 지원할 수 있는 확장 가능한 구조를 가질 수 있다.

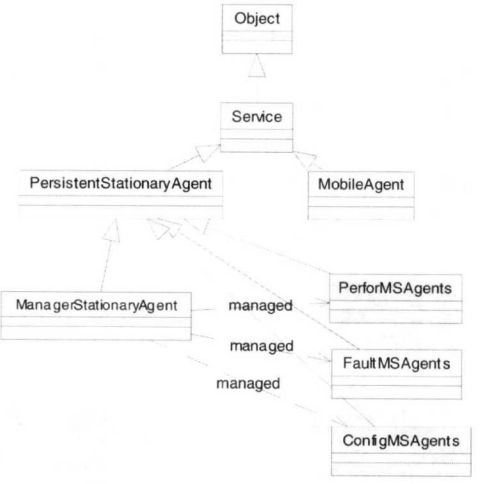


그림 8. 고정 에이전트 클래스 계층도

위의 계층도에서 보이는 ManagerStationaryAgent는 JMX에 있어서 MBean Server의 역할을 한다.

또한 모든 고정 에이전트는 Grasshopper 플랫폼의 영속성(Persistent Service)서비스[19]를 이용하여 Network Element에 대한 자원관리를 할 수 있다.

표 2. MIB 값의 재분류

Agents Names	MIB Variables
PerforMSAgents	SnmpInPkts, snmpOutPkts, sysUpTime, ipDefaultTTL, ifInDiscards, ifOutDiscards, iFInErrors, ifOutErrors, ifInOctets, ifOutOctets, ifInUcastPkts, ifOutUcastPkts, ifInNUcastPkts, ifOutNUcastPkts, ifInUnknownProtos, ifOutQLen, ipInReceives, ipInHdrErrors, ipForwDatagrams, ipInUnknownProtos, ipInAddrErrors, ipInDiscardsm upInDelivers, ipOutDiscards, ipOutNoRoutes, ipRoutingDiscards, ipReasmReqds, ipReasmOKs, ipReasmFails, ipFragOKs, ipFragFails, ipFragCreates
FaultMSAgents	SysObjectID, sysServices, sysUptime, upInHdrErrors, ipInAddrErrors, upReasmFails, ipInReceives, ipForwDatagrams, ipInDelivers, upOutRequest, ipOutDiscards, ipOutNoRoutes, ipRoutingDiscards, ipReasmReqds, ipReasmOKs, ipReasmFails, ipFragOKs, ipFragFails, ipFragCreates
ConfigMSAgents	SysDescr, sysLocation, sysName, ifDescr, ifType, ifMtu, ifSpeed, ifAdminStatus, ipFowarding, ipAddrTable, ipRouteTable

Ⅳ. 제안하는 플랫폼 구조의 설계 및 구현

세 번째 항목에서 설명했던 것처럼 전체적인 플랫폼은 이동 에이전트와 고정 에이전트의 2개의 큰 분류로 이루어진다.

전체적인 플랫폼의 그림은 아래와 나와있는 그림 9과 같으며 이제부터 각 컴포넌트의 설계에 대해서 설명하도록 하겠다.

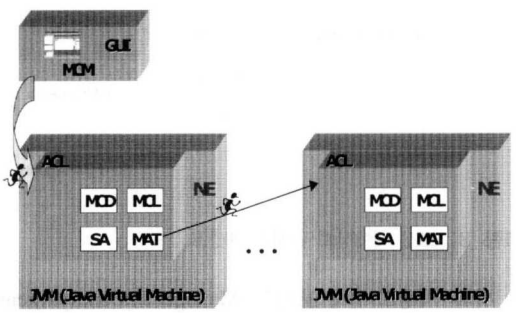


그림 9. 제안하는 플랫폼의 구조

위의 그림에서 보는 바와 같이 전체적인 플랫폼은 관리자(manager)와 관리 대상 노드(Network Elements)로 구분이 되어진다. 관리자에 의해서 생성되는 이동 에이전트는 목적지 노드로 이동하게 되며 주어진 임무를 수행하게 된다.

4.1 이동 코드 관리자 (MCM : Mobile Code Manager)

이동 코드 관리자는 코드 저장소(Code Repository)를 참조하여 관리자의 요청에 의해 이동 에이전트를 생성한다. 생성된 이동 에이전트는 이동 코드 관리자에 의 목적지 노드로 보내지게 된다. 이동 코드 관리자의 구성은 다음 그림과 같다.

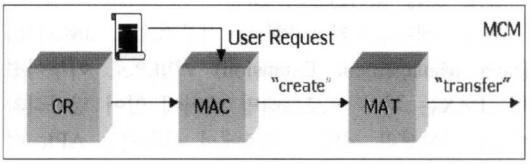


그림 10. 이동 코드 관리자의 구성

⊙ 코드 저장소 (CR : Code Repository) : 이동 코드 관리자에 의해서 이동 에이전트가 생성될 때 관리자의 요구에 따라 망관리 기능에 부합하는 (성능, 계정, 구성) 코드를 참조하게 된다. 코드 저장소에는 성능, 계정, 구성 요소 관리에 필요한 함수와 변수들이 정의되어있다.

⊙ 이동 에이전트 생성자 (MAC : Mobile Agents Creator) : 코드 저장소에 있는 코드와 망관리자의 필요에 의해 첨가된 코드를 이용하여 이동 에이전트를 생성한다.

⊙ 이동 에이전트 전달자 (MAT : Mobile Agents Transfer) : 생성된 이동 에이전트를 목적지 노드로 이동 시키는 역할을 한다. (전달의 기능은 Grasshopper 플랫폼에 의존한다.)

4.2 동적 클래스 로더 (ACL : Active Class Loader)

동적 클래스 로더는 Network Element에 존재하며 실질적인 망관리 작업을 이동 에이전트와 수행하는 모듈이다. (그림 11)

⊙ 이동 코드 데몬 (MCD : Mobile Code Daemon) : 이동 에이전트가 목적지 노드에 도착하게 되면 이동 코드 데몬이 도착 여부를 체크하여 이동 클래스 로더 (MCL)에게 통보하여 준다.

⊙ 이동 클래스 로더 (MCL : Mobile Class Loader) : 목적지 노드에 도착한 이동 에이전트의 정보를 읽어들인다. 읽어들인 정보에서 고정 에이전트 활성화에 필요한 인자값만 추출하여 고정 에이전트로 인자값을 전달한다.

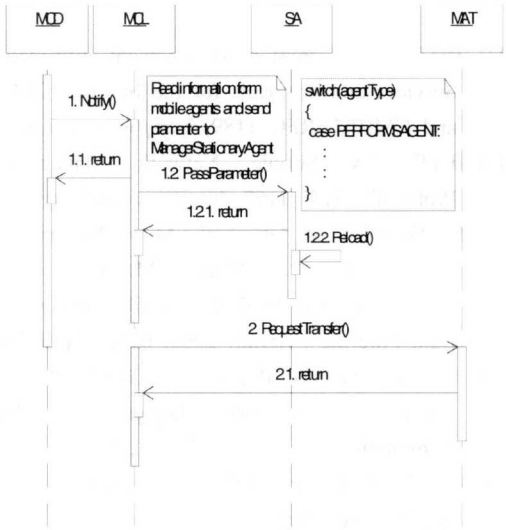


그림 11. 동적 클래스 로더의 순차 다이어그램

⊙ 이동 에이전트 전달자 (MAT : Mobile Agents Trasfer) : 이동 에이전트를 관리자에게 다시 되돌려 보내거나 다른 목적지로 보내는 역할을 한다.

⊙ 고정 에이전트 (SA : Stationary Agents) : 실제적인 망관리 요소이다. Grasshopper 플랫폼의 영속성 서비스를 이용해서 Network Element의 부하를 줄이며 MIB값들을 관리한다.

V. 제안구조 평가

제안된 이동 에이전트 플랫폼을 평가하기 위해서 이동 에이전트의 코드 크기와 노드의 고정 에이전트가 점유하는 메모리의 크기를 비교하였다.

시뮬레이션 환경은 다음과 같다.

- OS : Windows 2000 Professional
- PC : Pentium Ⅲ 733Mhz RAM 512M
- Tool : JDK 1.2.2, Grasshopper Platform 2.2.2

그림 6의 계층도에서 보이는 것과 같이 분류되는 이동 에이전트는 자신에게 주어진 임무를 수행하기에 최적화된 코드라 할 수 있다. 이렇게 클래스화된 이동 에이전트의 코드 사이즈는 Java 클래스 파일이기 때문에 불필요한 함수와 변수를 줄이고 기능적 관리 영역(Functional Management Area) 분류에 따라 컴파일하게 되면 코드 크기를 최소화, 최적화할 수 있게 된다.

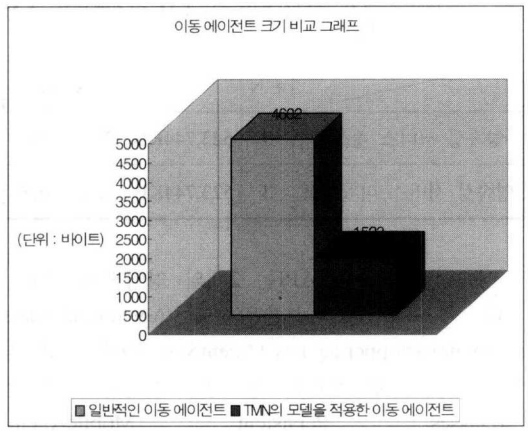


그림 12. 이동 에이전트 코드 크기 비교 그래프

왼쪽에 보이는 막대 그래프가 기능적 관리 영역 분류를 적요하지 않은 일반적인 이동 에이전트의 평균 코드 크기이며 오른쪽에 보이는 막대 그래프가 기능적 관리 영역 분류를 적용시켜 디자인한 이동 에이전트의 평균 크기를 나타내는 것이다. 그래프에서 보여주는 수치와 같이 기능적 관리 분류에 의해 재설계된 이동 에이전트의 코드 크기가 약 53% 정도 감소한 것을 볼 수 있다. 코드 크기가 작아진 이동 에이전트는 네트워크를 통해 목적지 노드로 혹은 다시 관리자에게 이동하는데 불필요한 네트워크 이동 트래픽이 감소하는 효과를 얻을 수 있다.

또한 고정 에이전트에서는 Grasshopeer 플랫폼이 제공하는 영속성(Persistent service)를 이용하여 망관리 플랫폼이 노드에서 차지하는 비율을 최소화할 수 있도록 고려하였다. 영속성 서비스는 Grasshopper agencies의 중요한 기능중에 하나이다. 영속성 서비스의 목적은 현재 Agency에서 실행되고 있는 모든 에이전트의 정보와 모든 Place의 실행 정보를 저장하는 것이다. 영속성 서비스는 현재 수행되고 있는 에이전트 중에서 필요가 없다고 판단되는 에이전트를 잠시 메모리에서 해제할 수 있으며 다시 해제된 에이전트가 필요하다고 판단되어질 때 메모리에 재할당을 할 수 있다. Network

Element의 주된 작업은 패킷 전달을 위한 라우팅이기 때문에 망 요소의 할당된 제한적인 리소스를 관리하는 것은 중요한 작업이라고 할 수 있겠다. 표 3에서는 메모리 측정과 측정에 사용된 고정 에이전트의 수에 대해서 나타내었다.

표 3. 메모리 차이 비교표
(#N : 고정 에이전트의 수, T.M : 총메모리, A.M : 가용메모리)

	# N	T. M	A. M
영속성 서비스 활성화	3	523,744KB	231,054KB
영속성 서비스 비활성화	3	523,744KB	224,529KB

영속석 서비스의 API는 2개의 영역으로 나뉘어진다. 한 부분은 Agency(interface de.ikv.grasshopper.agency.IAgentSystem)에 의한 영역이고 다른 한 부분은 영속성을 지원하는 에이전트(classes Persistent MobileAgent, PersistentStationary Agent) 이다.

Ⅵ. 결론 및 향후 연구과제

이동 에이전트를 망관리에 적용하게 되면 우리는 기존의 망관리 시스템보다 보다 효율적인 시스템을 얻을 수 있다. 그러나 아직까지 많은 시스템에 이동 에이전트 시스템을 적용시키지 못하고 있는 이유는 JVM 자체의 무거움과 자바 플랫폼 의존성 때문이다. 그러나 이런 문제들은 Java-Chip 이나 임베디드(Embedded) 시스템[20] 등을 통하여 해결할 수 있을 것이다.

본 논문에서 제안한 이동 에이전트 플랫폼은 TMN의 정보 구조(Information Architecture)를 바탕으로 한 OSI 관리 기능 분류를 이용하여 새롭게 에이전트들을 정의하였다. 또한 우리는 이동 에이전트의 코드를 최소화하여 에이전트의 이동시 발생할 수 있는 네트워크의 트래픽을 줄일 수 있는 방안에 대해서 연구하였고 JMX기반의 고정 에이전트에 영속성(Persistent) 서비스를 적용하여 망 요소의 효율적인 리소스 관리 방안에 대해서 연구하였다.

지금까지 우리의 연구는 주로 이동 에이전트를 이용한 일반적인 이동 플랫폼에 대한 연구였다. 앞으로 계속 연구되어야 할 문제는 특정 네트워크 영역에 과연 몇 개의 이동 에이전트를 적용해야 하는가의 효율성 문제와 단순히 이동 노드 지정(IP 주소 지정에 의한)에 의한 이동 가능한 영역을 지정해 주는 방식을 벗어나 이동 에이전트의 자율적인 이동성 보장을 위한 이동 경로 설정 알고리즘 문제 등이라고 할 수 있을 것이다.

참 고 문 헌

[1] IETF "The Common Management Information Services and Protocols for the Internet (CMOT and CMIP)", RFC 1189, http://www.ieft.org

[2] IETF, "A Simple Network Management Protocol", RFC 1157, http://www.ietf.org

[3] D. Gavalas, M Ghanbari, M. O'Mahony, D. Greenwood, "Emabling Mobile Agents Technology for Intelligent Bulk Management Data Filtering", *NOMS* 2000, 623p, April 2000

[4] A. Bieszczad, B. Pagur다, T. White, "Mobile Agents for Network Management", *IEEE* Communications Survey, Vol 1, No. 1

[5] CORBA(Common Object Request Broker Architecture), http://www.omg.org

[6] Java Remote Method Invocation (RMI), http://java.sun.com/products/jdk/rmi/

[7] B. pagurek, Y. Wang, and T. White "Integration of Mobile agents with SNMP : Why and How", *NOMS* 2000

[8] MIAMI project, "Mobile Agents for Network Management", *IEEE* Communications Survey, Vol 1, No 1

[9] The ACTSproject, http://www.ee.surrey.ac.uk/CCSR/ACTS/Miami/grasshopper.html

[10] The MIAMI project at UCL, http://www.ee.ucl.ac.uk/~dgriffin/miami/

[11] TMN's Information Architecture, http://snmp.cs.utwente.nl/tutorials/tmn/index-15.html

[12] IBM Aglets, http://www.trl.ibm.com/aglets/

[13] IKV++ Grasshopper 2, http://www.grasshopper.de/index.html

[14] ObjectSapce Voyager, http://www.objectspace.com/products/voyager/

[15] MASIF, http://www.det.ua.pt/Projects/difference/work/D7/d7chap4.html

[16] Java Management Extensions(JMX), http://java.sun.com/products/JavaManagement/

[17] Mi-Young Kang, Sung Kim, Cheul-Young

Kim, Ji-Seung Nam, "The design and implementation of functional class for real-time network management based on java", *Korea information science society*, Vol. 28, No 1, April 2001

[18] Web-based Enterprise Management, http://www.dmtf.org/standards/standard_wbem.php

[19] Grasshopper Service, http://www.grasshopper.de/

[20] Embeded Java, http://java.sun.com/products/embeddedjava/

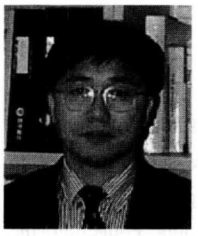
이 정 환(Jung Hwan Lee) 학생회원

2001년 : 경희대학교 전자계산공학과 졸업 (학사)

2001년~현재 : 경희대학교 전자계산공학과 석사과정

<주관심 분야> Network Management System, Active Network, Distributed Processing System

홍 충 선(Choong Seon Hong) 정회원



1983년 : 경희대학교 전자공학과 졸업 (학사)

1985년 : 경희대학교 전자공학과 (공학석사)

1996년 : Keio University, Department of Information and Computer Science (공학박사)

1988년~1999년 : 한국통신 통신망 연구소 선임 연구원/ 네트워킹연구실장

1999년~현재 : 경희대학교 전자정보학부 조교수

<주관심 분야> 인터넷 서비스 및 망 관리 구조, 분산 컴포넌트관리, IP 프로토콜, 멀티미디어 스트리밍 등