Mobile Agents를 이용한 효율적인 망관리 구조

이정환[0] 홍충선
경희대학교 전자정보학부
nowsys@networking.kyunghee.ac.kr[0] cshong@khu.ac.kr

An Efficient Mobile Agents Platform Architecture for Network Management

Jung-Hwan Lee[0] Choong-Seon Hong
School of Electronics and Information, Kyung Hee Univ.

요 약

기존의 단순한 객체를 이용한 중앙집중형 망관리 시스템이 가지는 단점들을 극복하고자 분산객체를 이용하는 Dynamic Object 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 MAs(Mobile Agents) 기반의 플랫폼이 제시되었으나 MAs 사용으로 추가적으로 발생할 수 있는 트래픽의 문제, NE(Network Element)의 리소스 관리 문제 등을 고려해야 한다.

본 논문의 주제는 바로 이 MAs를 이용한 망관리를 하는데 있어 보다 더 효율적인 구조를 연구했으며 이러한 장점들을 바탕으로 MAs가 이동할 때 발생할 수 있는 트래픽의 최소화 방안, NE의 효율적인 리소스 관리 및 업무 수행 능력의 향상을 위해서 TMN의 Information Architecture를 사용하여 에이전트를 설계하였다.

1. 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 문제점들을 해결하기 위해 망을 지속적으로 관리하고 모니터링 및 분석을 해주는 것은 이제 필수적인 작업이 되었다. ISO에서는 CMIS/CMIP(Common Management Information Services/Common Management Information Protocol)[1]을 IETF에서는 SNMP(Simple Network Management Protocol)[2]을 만들었다. 이들은 전형적인 Client / Server 형태의 중앙 집중화 된 구조이다. 관리자와 에이전트로 구분되어지는 이 구조는 비교적 구현이 쉽고 단순한 구조를 가지고 있어 많은 NE(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 Sub-Network를 실시간으로 관리하고 모니터링하기 어렵다는 문제점을 가지고 있다[3]. 이러한 중앙 집중화 된 관리구조의 문제를 해결하기 위해서 MAS(Multi-Agent System)[4]를 지원하는 분산 구조(Distributed Architecture)의 형태가 제안되었다.

여러 개의 분산된 객체들이 각각의 에이전트에 고정되어 있는 형태가 바로 CORBA 혹은 Java-RMI를 이용한 Static Object 플랫폼이며, 이와는 상반되는 것이 바로 MAs(Mobile Agents)를 이용한 Dynamic Object 플랫폼이라 할 수 있겠다. Dynamic Object 플랫폼이 Static Object 플랫폼에 비해서 가질 수 있는 장점은 유연성과 확장성이라고 할 수 있을 것이다. 이 논문에서는 Dynamic Object 플랫폼인 MAs가 가지는 장점들을 활용하고 MAs가 가질 수 있는 단점, 즉 MAs가 이동할 때 발생하는 추가적인 트래픽의 발생 문제 그리고 두 번째 NE의 효율적인 리소스 관리에 대해서 연구하였다. 논문의 구성은 다음과 같다. 제 2장에서는 망관리를 위해 제안되었던 관련 연구에 대해서 알아보고, 제 3장에서는 Mobile Agents와 Stationary Agents의 설계에 대해서 기술하며, 제 4장에서는 제안된 사항을 바탕으로 한 새로운 플랫폼의 설계 및 구현에 대해서 기술하며 그리고 마지막 제 5장에서는 결론과 앞으로의 과제에 대해서 설명한다.

2. Mobile Agents를 이용한 망관리 플랫폼

MAs를 망관리에 사용하면 결함 관리, 계정 관리, 구성 관리, 성능 관리, 보안 관리의 망관리 기능적 분류 요소들을 사용하는데 망의 불필요한 트래픽을 줄일 수 있고, NE의 불필요한 자원 사용을 줄일 수 있으며 이종 환경을 지원하는 등의 장점을 가질 수 있다.

표1은 MAs를 이용한 망관리에 대해서 가질 수 있는 장단점에 대해서 정리하였다.

항 목	내 용
장 점	효율성 극대화 MAs의 이동에 따른 NE의 저장공간 절약 이종환경에 대한 지원 확장성 쉬운 소프트웨어 업그레이드
단 점	MAs의 이동 시 추가적인 트래픽 발생 MAs의 이동 가능한 도메인 범위 지정문제

표 1. MAs 기반의 망관리 시스템의 장단점

2.1 망관리에 적용된 플랫폼 구조

일반적인 구조는 그림 1과 같으면 망관리자에 의해 생성

되는 MAs는 관리를 필요로 하는 목적지 노드(Target Node)로 이동하게 된다.

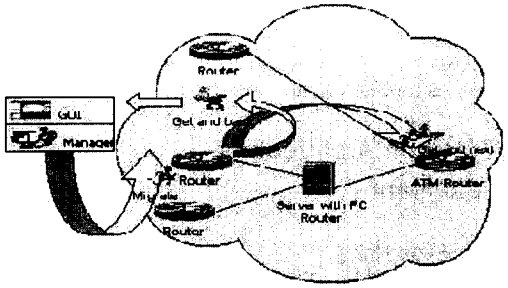

그림 1. 망관리에 MAs를 이용한 일반적인 구조

그림 1과 같이 단순하게 GnG, GnB(Get and Go, Get and Back)[3]의 동작들로만 정의된 구조로는 MAs의 관리, 망의 성능 관리, 데이터들의 효율적 관리에 문제점이 있기 때문에 이러한 점을 보완하기 위해 새로운 구조와 알고리즘을 가지는 플랫폼들이 제안되었다.

2.2 MIAMI 프로젝트

MIAMI(Mobile Intelligent Agents in the Management of the Information Infrastructure)[5]는 망관리와 서비스 관리에서 MAs의 사용 가능성과 영향을 평가하기 위한 프로젝트이다. 이 연구에서는 앞에서 언급한 MAs 플랫폼의 문제점들을 해결하기 위해 호스트에 제한된 이동성을 가지는 소프트웨어 에이전트를 배치시키고 AVP(Active Virtual Pipe)[6]를 통해 NE의 동적이고 효율적인 MAs 플랫폼(그림 2)을 제시하였다.

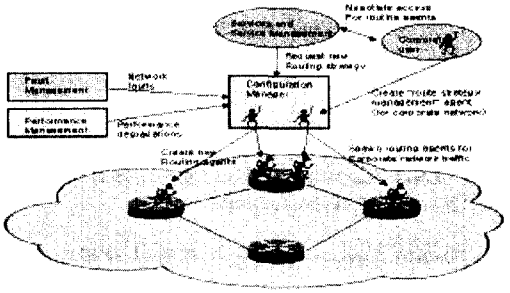

그림 2. MIAMI 플랫폼의 구조

3. 각 Agent 설계 및 구현

MAs 플랫폼에서 Agents는 크게 두 개의 분류로 나누어지는데 첫 번째는 Mobile Agents이고 두 번째는 Stationary Agents이다. Mobile Agents는 이동성을 가지고 목적지 노드를 방문해 필요한 정보를 수집하는 것이고 Stationary Agents는 MAs와 반대로 동적으로 다른 지역으로 이동할 수 없는 반면에 특정한 한 지역을 관리하는 Agents이다. 모든 Agents는 효율적인 관리를 위해 기본적으로 TMN의 Information Architecture를 사용하였고 OSI의 관리 정보 모델[7]을 사용하여 각각 장애관리, 구성관리, 성능관리, 계정관리, 보안관리의 분류로 나뉘어진다.

3.1 Mobile Agents 설계

MAs의 플랫폼은 Aglets[8], Grasshopper[9], Voyager[10] 등 각 Vendor마다 특징을 가지는 다양한 플랫폼이 있지만 MIAMI 프로젝트의 결과 보고서에서 추천한 IKV++사의 자바 기반의 Grasshopper 플랫폼을 사용하였다. 기본적으로 MAs는 속성과 행위를 가지는 객체이므로 불필요한 행위와 속성을 줄여 추가적으로 발생할 수 있는 트래픽의 양을 줄이고자 MAs를 위에서 언급한 5개의 분류항목으로 분류하였고 그 중에서도 MAs가 유용하게 적용될 수 있는 3가지 분류(그림 3)로 다시 분류하였다 .

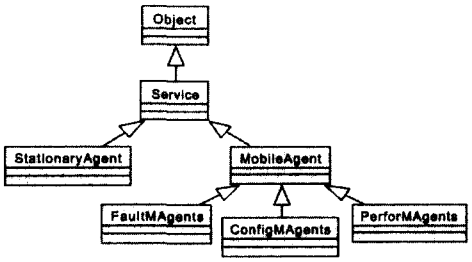

그림 3. MAs의 class 계층도

이렇게 분류된 MAs의 코드 크기를 비교하자면 아래에 나와있는 그림 4와 같으며 이러한 이유는 기본적으로 MAs의 코드 자체가 Java 클래스 파일이기 때문이며 관리 기능에 따라 분류를 나누었기 때문에 MAs의 불필요한 속성과 행위를 줄일 수 있기 때문에 가능했던 것이다. 일반적으로 SNMP PDU가 418Byte 미만이지만 필요한 데이터를 얻기 위해 빈번히 발생하는 것을 고려할 때 MAs의 Byte 크기가 결코 큰 것이 아님을 알 수 있다.

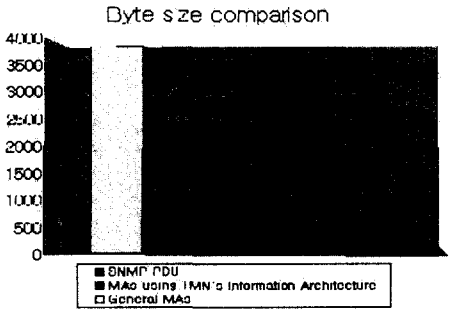

그림 4. Byte 크기 비교 그래프 (단위 : Byte)

3.2 Stationary Agents 설계

SAs(Stationary Agents)의 구현은 MAs와 마찬가지로 Grasshopper 플랫폼과 JMX(Java Management Extention)[11]을 사용하였다. 기본적으로 Grasshopper 플랫폼이 제공해주는 SAs에 JMX SNMP API를 사용하였다.

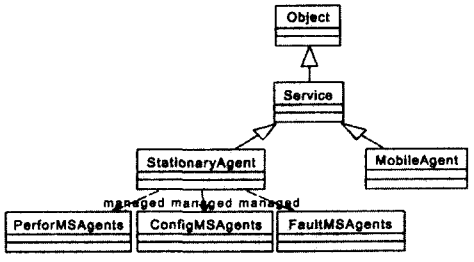

그림 5. SAs의 class 계층도

JMX의 구조는 기본적으로 3 레벨이며 관리, 에이전트, 분산 서비스로 구분되어 진다. 관리 레벨은 EJB(Enterprise Java Beans)[12] 기반의 MBean(Managed Bean)의 형태로 이루어지며 그림 6에서와 같이 TMN의 Information Architecture와 흡사하다. 따라서 OSI 관리 기능을 기준으로 하여 같은 범주에 있는 MIB 인자들을 MBean의 형태로 그림 5와 같이 SAs가 관리하게 된다. 이렇게 만들어지는 MBean은 SAs에 의해서 관리가 쉽고 SNMP 폴링에 대한 response에 대해서 성능이 향상된다[13]. SAs는 MAs를 지원하는 동시에 MBean Server의 역할을 하기도 한다.

그림 6. TMN의 Information Architecture

또한 Grasshopper 플랫폼은 영속성(Persistence) 서비스를 제공하여 현재 NE에서 서비스 되고 있는 에이전트에 대해서 모든 실행 가능한 정보를 영구적으로 저장한다. 즉 현재 필요성이 떨어지는 SAs을 메모리에서 동적으로 해제와 그리고 재할당이 가능하다는 것이다. NE의 주 작업은 패킷 라우팅이기 때문에 부과적인 리소스 할당을 줄이는 것은 반드시 필요하다. 표 2를 보면 Grasshopper 플랫폼의 영속성 서비스를 이용한 Mbean의 수와 이에 대한 메모리 측정이 나와있다. 영속성(Persistence) 서비스를 위한 클래스는 de.ikv.grasshopper.agent 패키지 안에 정의되어 있으며 해당 클래스는 PersistentStationaryAgent이다.

	# N	T. M	A. M
MBean with persistence service	3	523,744KB	231,054KB
MBean (no service)	3	523,744KB	224,529KB

표 2. 메모리 사용량 비교
(# N : MBean의 수, T.M : 총 메모리, A.M : 가용 메모리)

4. 결론 및 향후 과제

망관리에 MAs 플랫폼을 적용하게 되면 다른 어떤 망관리 플랫폼보다 유연하고 확장가능한 구조를 가질 수 있다. 그러나 아직까지 MAs 플랫폼이 실제로 대부분의 망관리에 적용이 되지 않는 이유는 JVM(Java Virtual Machine) [14]자체의 무거움과 전체적인 플랫폼 의존성 등의 문제가 있으나 JVM 자체가 내장형(embedded)[15] 시스템으로 개발되고 있는 것을 생각해 보면 그러한 문제는 Java Chip등을 통하여 해결되어질 수 있는 과제라고 생각된다.

본 논문에서 제안한 MAs 플랫폼은 TMN의 Information Architecture를 바탕으로 한 OSI 관리 기능별 분류를 사용하여 MAs의 코드 크기를 최소화하여 추가적으로 발생할 수 있는 네트워크 트래픽을 최소화하는 방안과 EJB기반의 MBean에 영속성(Persistence) 서비스를 적용하여 NE의 리소리 관리 차원에서 적은 비용이 들 수 있는 방안을 연구하였다.

지금까지의 연구는 MAs를 이용한 전반적인 플랫폼에 대한 연구에 대해서 기술하였다. 앞으로 연구할 방향은 아직은 에이전트 별로 설계되고 구현된 시스템을 통합하는 일과 MAs 플랫폼 관리 영역(Managed Domain)에 과연 몇 개의 MAs를 적용해야 하는 MAs 플랫폼 자체에 대한 효율성 검증과 이동 경로 설정에 대한 알고리즘에 대한 연구도 필요하다.

5. 참고문헌

[1] IETF, "The Common Management Information Services and Protocols fot the Internet(CMOT and CMIP)", RFC 1189, http://www.ietf.org
[2] IETF, "A Simple Network Management Protocol", RFC 1157, http://www.ietf.org
[3] D. Gavalas, M Ghanbari, M. O'Mahony, D.Greenwood, "Enabling Mobile Agent Technology for Intelligent Bulk Management Data Filtering", NOMS 2000, 623p, April 2000
[4] A. Bieszczad, B. Pagurek, T. White, "Mobile Agents for NEtwork Management", IEEE Communications Survey, Vol 1, No. 1, http:///www.comsoc.org/pubs/surveys, 4Q1998
[5] MAIMI project, "Mobile Agent Platform Assessment Report", http://www.fokus.gmd.de/research/cc/ecco/climate/ap-documents/miami-agplatf.pdf
[6] George Eleftheriou, Alex Galis, "Mobile Intelligent Agents for Network Management Systems", http://www.ee.ucl.ac.uk/~pants/projects/miami/htm monstrations.html
[7] TMN's Information Architecture, http://snmp.cs.utwente.nl/tutorials/tmn/index-15.html
[8] IBM Aglets. http://www.trl.ibm.com/aglets/
[9] IKV++ Grasshopper 2, http://www.grasshopper.de/index.html
[10] ObjectSpace Voyager, http://www.objectspace.com/products/voyager/
[11] JMX, http://java.sun.com/products/JavaManagement/
[12] EJB, http://java.sun.com/products/ejb/
[13] 강미영, 김성, 김철용, 남지승, "실시간 망 관리를 위한 JAVA 기반의 기능별 클래스 설계 및 구현", 한국정보과학회 봄 학술 발표논문집 Vol. 28, No. 1, 4월 20001년
[14] Java JVM, http://java.sun.com/
[15] Embeded Java, http://java.sun.com/products/embeddedjava/