이동 에이전트 기술의 대등 관계 패러다임과 응용 모델

김평중* · 진성일**

1. 서 론

정보화 사회가 진전됨에 따라 사용자의 작업을 대신해주는 에이전트의 필요성이 증가하고 있다. 인터넷의 대중화와 함께 정보의 홍수 속에서 사용자 스스로 원하는 물품에 대한 최신의 정보를 정확하고 빠른 시간에 얻는 것은 매우 어려운 상황이다. 에이전트는 사용자의 요구에 따라 물품 정보, 여행 정보, 주식 정보 등을 인터넷에서 검색하여 보여주거나 복잡한 일상 생활을 시기 적절하고, 효율적으로 대신 처리하여 준다. 특히, 이동 에이전트는 인터넷 환경에서 분산 응용 프로그램 작성에 용이한 형태를 제공한다.

이동 에이전트(mobile agent)란 에이전트가 수행을 시작한 시스템에 묶여있지 않는 에이전트이다. 고정 에이전트(stationary agent)에 비교해 보면 서로 다른 시간에 서로 다른 시스템에서 수행될 수 있는 경우이다. 이동 에이전트는 이질적인 망(heterogeneous network)에서 자신의 제어로 호스트 사이를 이주(migration)하고, 각 호스트의 다른 에이전트와 상호 동작하거나 자원을 이용하여 맡겨진 임무를 수행하고, 수행이 끝났을 경우 고향(home)으로 되돌아온다.

에이전트가 인터넷을 돌아다니며 수행되기 위해서 코드뿐만 아니라 상태도 이동되어야 한다. 코드는 이동될 모든 호스트에 똑같은 형식으로 동작되어야 하고, 직접 해석(interpretation)되거나 다시 컴파일(recompilation)하지 않고 수행될 수 있는 이식성있는 중간 언어(portable intermediate interpreter-based language)가 되어야 한다. 상태는 지속성(persistency)이 있어야 한다. 이동 코드의 수행 위치, 중간 결과 등을 계속 유지해야 하기 때문이다. 기존에 수행되었던 곳에서부터 다시 수행되도록 하기 위하여 지속성 상태가 코드의 부분으로서 표현되거나 부호화된 스택(encoded stack)으로 전송되어야 한다.

통신 형태에 따라 분류하면, 이동 에이전트는 클라이언트-서버 모델(client-server model) 보다는 오히려 대등 관계 모델(peer-to-peer model)이다. 각 에이전트는 경우에 따라 클라이언트 또는 서버 역할을 수행할 수 있다. 예를 들면 사용자의 요구를 담은 에이전트가 이것을 처리해줄 수 있는 서버 시스템으로 직접 가서 서비스를 받은 후 그 수행 결과를 클라이언트에게 다시 가져오는 형태이다. 이 때 사용자 요구를 담은 에이전트는 클라이언트-서버 모델의 클라이언트이든 서버이든 특정 역할로만 동작되지 않고, 단지 대등 관계 형태로서 동작한다.

본 논문에서는 2장에서 클라이언트-서버 모델과 대등 관계 모델에 대한 paradigm을 비교 설명하고, 3장에서는 대표적인 자바 이동 에이전트 시스템을 분석하고, 4장에서는 이동 에이전트 기술

*충북도립옥천대학 컴퓨터정보과 교수
**충남대학교 컴퓨터과학과/소프트웨어 연구센터 교수

의 응용을 설명하고, 마지막으로 5장에서 결론 및 향후 연구 방향을 기술한다.

2. 클라이언트-서버 모델과 대등 관계 모델

최근 들어 많은 연구자들은 "클라이언트(client)와 서버(server)간에 메시지가 아니라 수행할 수 있는 프로그램(executable program)을 전송함으로써 클라이언트-서버 컴퓨팅을 수행한다"는 아이디어를 제시하였다. 기존의 스크립트 프로그램의 원격 급송(remote dispatch of script program)이나 일괄작업의 원격 처리(remote submission of batch job) 등의 개념이 여기에 포함된다. 이동 에이전트 기반 컴퓨팅(mobile agent-based computing)은 이 기법의 확장으로 볼 수 있다[1].

네트워크 컴퓨팅(network computing)에서 이동 에이전트는 많은 장점을 제공한다. 이동 에이전트는 정보자원(information resource)이 존재하는 컴퓨터에 더 가까이 접근하여 작업을 수행한다. 이동 에이전트 패러다임(mobile agent paradigm)은 클라이언트-서버 패러다임의 확장으로 볼 수 있다. 이동 에이전트가 전통적인 클라이언트-서버 구조에서 얻을 수 없는 새로운 기능을 제공하는 것은 아니다. 그러나, 새로운 기능을 더 쉽게 구현할 수 있도록 한다. 근본적인 장점은 서버가 제공하는 서비스와 그 서비스를 사용하는 방법간에 한 계층의 추상화(a layer of abstraction)를 제공한다[2].

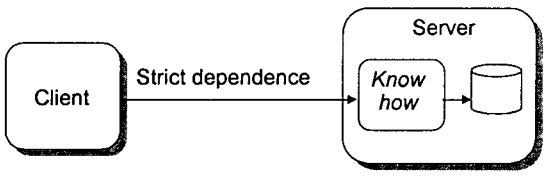

그림 1. 클라이언트-서버 패러다임

클라이언트-서버 패러다임(client-server paradigm)에서 통신하는 양측은 정확히 정의되고 고정된 역할(fixed and well-defined role)을 갖는다. <그림 1>과 같이 서버는 데이터베이스와 같은 자원에 접근을 제공하는 서비스의 집합을 정의하고, 클라이언트는 그 서비스를 사용한다. 서버는 서비스를 수행하고, 노하우(know-how)를 갖고 있다. 이 모델은 엄격한 의존관계(strict dependence)를 갖는다. 클라이언트는 서버가 제공하는 서비스들만 이용할 수 있다. 클라이언트와 서버간에 발생하는 통신 메커니즘은 메시지 전송 프로토콜을 통해서 이루어진다. 메시지 전송을 이용할 경우 네트워크 주소와 동기점(synchronization point)을 프로그래머가 결정해야 하기 때문에 매우 낮은 프로그래밍 수준에서 이루어 진다고 볼 수 있다[3].

RPC(Remote Procedure Call)는 프로그래머에게 매우 낮은 프로그래밍 수준의 부담을 없애기 위하여 Sun Microsystems사가 개발한 것이다. 여기서는 클라이언트가 자기 시스템의 함수 호출과 똑같은 방법으로 서버에서 수행되는 서비스를 요구할 수 있다. 이러한 서비스는 stub에 의해서 표현된다. stub는 RPC 시스템에 의해서 전송되는 템플릿 함수 호출(template function call)이다. 서버의 위치(location of server), 서비스의 시작(initiation of service), 결과의 전송transportation of results) 등은 클라이언트에게 투명하게 처리된다. 따라서 추상화(abstraction)의 수준을 한 단계 높임으로써 프로그래머는 복잡성의 부담을 줄일 수 있게 된다. Sun사의 RPC 이외의 다른 클라이어트-서버 구조로는 CORBA(Common Object Request Broker)와 DCE(Distributed Computing Environment) RPC가 있다. CORBA는 객체 재사용(object reuse), 상속(inheritance), 캡슐화(encap-

sulation) 등 객체지향 원리(object-oriented principle)를 클라이언트-서버 구조에 채택함으로써 보다 더 이용하기 쉽게 만든 시스템이다. DCE RPC는 복잡성을 줄이는 측면보다 보안(security)과 인증(authentication) 기능, 사용자 수준의 쓰레드(user-level thread)를 제공한다.

그러나, 분산 정보 관리(distributed information management) 시스템에서 클라이언트-서버 구조는 근본적인 취약점을 가지고 있다. 우선, 클라이언트가 요구하는 서비스를 서버가 정확히 갖고 있지 않은 경우 클라이언트는 자기의 목적을 달성하기 위하여 일련의 원격 호출(a series of remote call)을 해야한다. 결국 전체 지연시간(overall latency)을 증가시키고 불필요한 중간 데이터(wasteful intermediate information)를 발생시킨다. 만약 서버가 좀더 특별한 서비스(more specialized service)를 통해 이 문제를 해결한다고 가정하면, 클라이언트의 수가 증가할 때 서버 당 요구된 서비스(amount of services per server)를 제공하는데 한계가 있게 된다.

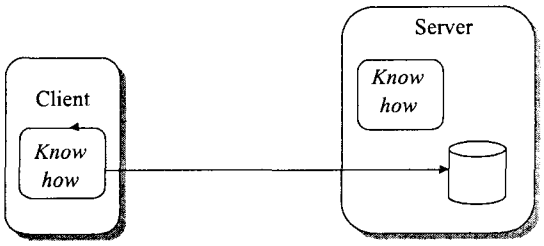

그림 2. 서브프로그래밍 패러다임

서브프로그래밍(subprogramming)은 code-on-demand 패러다임으로 불리고, 클라이언트-서버 구조의 문제점을 어느 정도 보완하고 있다. 그림 2에서 보는 바와 같이 클라이언트가 서브프로그램을 서비스가 위치한 서버시스템으로 보내 수행할 수 있다. 서브프로그램은 초기에 이주할(initial migration) 수 있으나 계속해서 다른 시스템으로 이동할 수는 없다. 따라서 서브프로그램은 엄격한 클라이언트-서버 관계를 지켜야하고 오랜 시간 동안 수행될 수 없는 제약점이 있다. 일반적으로 서브프로그램은 특정 클라이언트를 위해 쓰여지기 때문에 재사용(reusability)의 정도가 감소된다. Java applet과 servlet이 좋은 예이다.

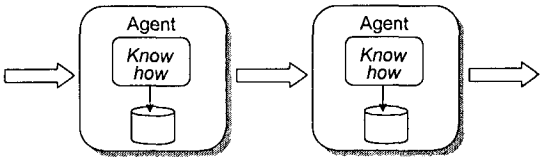

그림 3. 이동 에이전트 패러다임

그림 3에서와 같이 이동 에이전트 패러다임(mobile agent paradigm)은 클라이언트-서버 패러다임과 서브프로그래밍 패러다임의 문제점을 처리한다. 이동 에이전트의 대표적인 특징은 마음대로 이주할 능력(migration at will), 자율적인 행동 양식(autonomy in their action), 대등 관계 성격(peer-to-peer personality), 처리와 네트워크의 독립성(processing and network independence) 등이다.

3. 대표적인 자바 이동 에이전트 시스템

이동 에이전트 시스템(mobile agent system)은 이동 에이전트를 생성, 이동, 수행, 전송, 해석, 폐기 등 에이전트의 생명 주기(life cycle)를 관리할 수 있는 플랫폼이다. 이동 에이전트 시스템의 주요 구성요소는 에이전트를 프로그래밍하기 위한 언어(language), 에이전트를 설치하기 위한 라이브러리(library), 에이전트를 수행시키기 위한 인터프리터(interpreter), 에이전트를 전송하기 위한 프로토콜(transfer protocol), 및 에이전트를 보

호하기 위한 보안(security) 등이다. 이동 에이전트를 구현하는데 사용하는 프로그래밍 언어의 요구사항은 호스트에서 수행 환경(execution environment on host)을 제공하는 것뿐이다. 다양한 언어가 연구용 시스템이나 상용 초기 시스템으로 사용되어 왔다. DEC의 Obliq나 GeneralMagic의 Telescript는 이동 에이전트를 구현하기 위해 특별히 설계된 언어이다. Java, Dartmouth College의 AgentTcl, Perl5, Python 등은 범용 언어에다가 특정 라이브러리를 추가하여 이동 에이전트를 개발하는데 사용하고 있다[4]. 최근의 전반적인 경향은 Java 언어에 근거한 기반 구조(infrastructure)를 개발하는 것이 대세로 보인다.

Java는 상대적으로 젊은 언어이지만 인터넷과 인트라넷 응용을 개발하는 산업 표준(de facto standard) 언어로 이미 위치를 확고히 하고 있다. Java는 범용 객체지향 언어로서 클래스(class)를 객체지향 모델로 사용하고 있고, 구문은 C 언어와 C++ 언어와 유사하다. Java가 이동 에이전트를 위하여 특별히 설계된 것은 아니지만, 이동 에이전트 프로그래밍에 필요한 대부분의 기능을 갖고 있다. Java는 멀티쓰레드(multi-thread)를 지원하고 있고, Java 프로그램은 컴파일되어 Java 바이트 코드(byte codes)로 변환된다. Java 바이트 코드는 Java 가상 기계(JVM:Java Virtual Machine)의 인터프리터(interpreter)에서 수행될 이진 명령어(binary instruction)이다. Java 프로그램은 Java 가상 기계를 가진 어떤 플랫폼에서도 수행될 수 있기 때문에 고도의 이식성(high portability)을 갖는다. Java 라이브러리는 통신 절차를 효율적으로 지원하고 있다. 이렇게 해서Java는 이동 에이전트 시스템을 구현하는 언어로서 각광을 받게 되었고, 이 경우 대부분의 시스템들은 Java의 class loading 모델, Java1.1의 RMI(Remote Method Invocation), 객체 serialization 메커니즘, reflection 메커니즘 등을 사용하고 있다[5].

Java를 사용한 이동 에이전트 관련 시스템은 Agent Society 홈페이지(http://www.agent.org)에 현재 수많은 상용 에이전트 시스템과 연구용 에이전트 시스템의 목록에서 확인할 수 있다[6]. 이들 중 비교적 성숙된 시스템이고, 웹으로부터 다운로드 받을 수 있는 대표적인 상용 시스템인 IBM의 Aglets와 ObjectSpace의 Voyager, General Magic의 Odyssey 등을 비교 분석하였다.

3.1 IBM의 Aglets

Aglets Workbench(AWB)는 IBM사의 Tokyo Research Lab(TRL)에서 개발한 이동 에이전트 시스템이다. 실행 환경과 함께 이동 에이전트 프레임워크를 Java 클래스 라이브러리 형태로 제공하여 개발을 쉽게 하였다. 실행 환경은 Tahiti라 불리는 GUI(Graphic User Interface) 기반의 Java 응용으로 이동 에이전트 관리를 쉽게 하려는데 초점을 두었으며, 웹 페이지 내에 존재할 수 있는 실행 환경인 Fiji를 제공하고 있다. 이동 에이전트의 전송을 위한 프로토콜인 ATP(Agent Transfer Protocol)와 프레임워크인 J-AAPI(Java Aglets API)가 표준으로 채택되도록 노력하고 있다. J-AAPI는 하나의 Aglets을 초기화, 메시지 핸들링, 급송(dispatching), 회수(retracting), 활동화/비활동화(activation/deactivation), 복사(cloning), 철회(disposing) 등을 제공하는 메쏘드를 제공한다. 응용 개발자는 플랫폼 독립적인 Aglets을 프로그래밍할 수 있고, J-AAPI를 지원하는 어떤 호스트에서도 수행할 수 있다.

Aglets Object Model은 인터넷의 개방형 광역 네트워크에서 이동 에이전트 기술을 이용하는데 필요한 추상화(abstraction)와 행위(behaviour)의

집합으로 설계한다. 주요 추상화는 aglet, proxy, context, message, future reply, identifier 등이다 [7].

첫째, Aglets은 네트워크에서 호스트를 방문하는 이동 Java 객체이다. Aglets은 메시지에 응답할 능력을 갖고 있기 때문에 응답성(reactive)이 있고, 호스트에 도착한 후 자신의 쓰레드를 수행할 수 있기 때문에 자율성(autonomous)을 갖고 있다. 둘째, Proxy는 Aglets의 대리인(representative)으로서 직접 접근으로부터 Aglets을 보호하는 보호막으로서 동작한다. 또한 Aglets을 위해 Aglets의 실제 위치를 숨길 수 있는 위치 투명성(location transparency)을 제공한다. 셋째, Context는 Aglets의 작업공간(workspace)로서 호스트가 악성 Aglets으로부터 안전하게 수행할 수 있는 환경에서 Aglets을 유지하거나 관리하는 정적 객체이다. 컴퓨터 네트워크에서 하나의 노드는 여러 개의 서버를 운영할 수 있고, 하나의 서버는 여러 개의 Context를 주최할 수 있다. Context는 서버의 주소와 Context의 이름을 조합하여 명명되어야 한다. 넷째, Message는 Aglets들간에 교환되는 객체이다. Aglets들간에 동기 메시지 전송과 비동기 메시지 전송을 제공한다. 다섯째, Future Reply는 비동기적으로 메시지를 전송하고 후에 결과를 받는데 사용한다. 여섯째, Identifier는 각 Aglets과 밀결합되어 있다. Identifier는 유일하고 Aglets의 생존 동안에 변경될 수 없다.

그림 4는 Aglets 생명 주기 모델을 보여준다. Aglets 객체 모델이 제공하는 행위(behaviour)로부터 이동 에이전트의 삶과 죽음을 분석한다. Aglets이 생성되는 방법은 스크래치로부터 시작되거나(create) 기존의 Aglets으로부터 복사하는 방법(cloning)이 있다. create는 하나의 Context에서 하나의 Identifier를 지정받고, 그 Context로

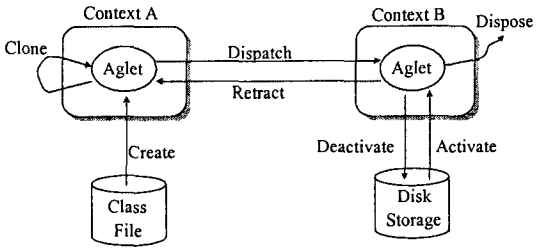

그림 4. Aglets의 생명 주기 모델

삽입되어 초기화된다. 성공적인 초기화가 끝나자마자 수행이 시작된다. Cloning은 동일한 Context에서 원래의 Aglets과 거의 같은 복사본을 생성한다. 차이점은 새로운 Identifier를 지정하고, 수행 쓰레드는 복사되지 않기 때문에 복사된 Aglets에서 수행을 재시작(restart)한다. Aglets의 수를 제어하기 위하여 Aglets을 파기할(disposal) 수 있다. disposal은 현재 Aglets 수행을 정지시키고 현재의 Context로부터 제거한다.

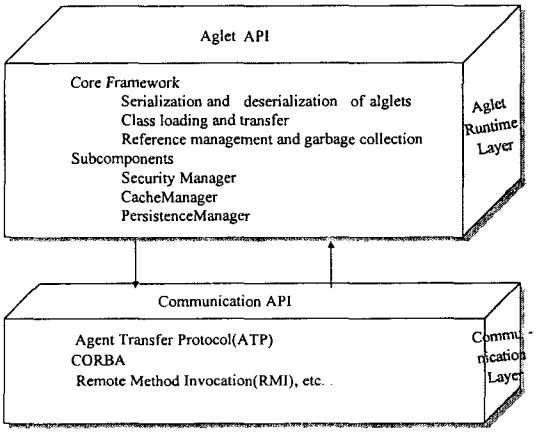

그림 5. Aglets의 API 구조

그림 5는 Aglets의 API 구조로서 Aglets Runtime 계층과 Communication 계층으로 구성되고, 각 계층의 기능들을 접근하기 위한 인터페이스를 정의한다. Aglets Runtime 계층은 Aglets이 생성되

고, 관리되고, 원격 호스트에 이주하는 등의 기본 기능을 제공하는 계층으로서 core framework과 여러 개의 subcomponent로 구성된다. 이러한 components는 interface나 abstract class로서 정의되고, 각 서버 개발자는 필요한 기능을 구현하여 runtime으로 플러그인 한다. 따라서 일반 Aglets 응용 프로그래머는 이러한 API를 사용할 수 없고, 보이지도 않는다. core framework는 Aglets 수행에 기본적인 매카니즘 즉, Class loading과 전송, Aglets의 serialization과 deserialization, Reference management와 garbage collection 등의 기능을 제공한다. subcomponent는 다양한 환경이나 요구 사항에 따라 확장성 있고, 맞춤성 있도록 설계되었다. 대표적인 subcomponent는 PersistenceManager, CacheManager, SecurityManager 등이 있다. Communication 계층은 Aglets runtime 계층과 달리 시스템간의 프로토콜에 따라 Aglets의 serialized data를 목적지로 전송하거나 받는 기능을 제공한다. 현재 Agent Transfer Protocol(ATP)를 기본 프로토콜로서 사용한다. ATP는 HTTP 프로토콜을 모델링한 것으로 이동 에이전트를 전송하는 응용 수준의 프로토콜이다.

3.2 ObjectSpace의 Voyager

Voyager는 ObjectSpace 사의 이동 에이전트 기능을 갖춘 ORB(Object Request Broker) 제품으로서 Java 프로그래머가 쉽고 빠르게 세련된 네트워크 응용 프로그래밍을 할 수 있도록 환경을 제공한다. Voyager는 실행 환경인 Voyager가 있고, 또 실행 환경은 처음부터 웹에서의 이용을 염두에 두고 만들어졌으며, GUI를 별도로 갖지 않는다. 실행 환경을 통하지 않고 직접 원격 객체의 레퍼런스(reference)를 이용할 수 있는 기능, 여러 가지 통신 방법 지원 등의 다양한 특성을 가지고 있다[8].

Voyager는 전통적인 분산 프로그래밍 기법(RPC)과 이동 에이전트 프로그래밍 기법을 솔기가 없이(seamless) 연결해주며, 이에 대한 특징은 다음과 같다. 첫째, 원격 객체를 구성하거나 전송하는데 Java 메시지 구조를 사용한다. 둘째, 몇 분만에 네트워크를 돌아다니며 일을 하는 이동 에이전트를 생성한다. 셋째, 이동 에이전트의 위치 투명성(location transparency) 제공한다.

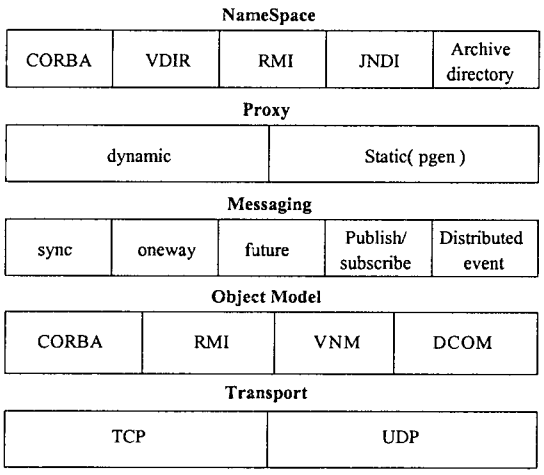

그림 6. Voyager의 구조

그림 6은 사용자 코드와 통신 프로토콜 및 메시징 프로토콜이 분리되는 Voyager의 전체 구조(universal architecture)를 보여준다. Voyager ORB는 대부분의 산업 표준(CORBA, DCOM 등)에 부합되고, 이에따라 간단한 접근 방법을 제공한다[9,10]. 통신 구조(communication architecture)는 Voyager 프로그램이 CORBA 프로그램, RMI (Remote Method Invocation) 프로그램, 그리고 DCOM(Distributed Component Object Model) 프로그램과 동시적이고 양방향 통신(simultaneous

bi-directional communication)을 할 수 있도록 제공한다. 메시지 계층은 위치와 객체 모델(location and object model)에 관계없이 synchronous, one-way, future 등 다양한 타입의 메시지를 전송할 수 있도록 한다. Voyager 이름 서비스(naming service)는 기존의 많은 상용 이름 서비스를 사용할 수 있도록 통일된 접근 방법을 제공한다. 하나의 API를 통하여 다양한 이름 서비스에 접근할 수 있다. Voyager 디렉토리 서비스(directory service)는 모든 클라이언트에게 공유되고 접근할 수 있도록 하나의 디렉토리(single directory)를 제공한다. 예를 들어 RMI 서버는 자신의 RMI 레지스트리 API를 이용하여 하나의 객체를 보편 디렉토리와 바인딩 할 수 있고, CORBA 클라이어트는 CORBA 명명 서비스 API를 이용하여 그 객체를 찾을 수 있다.

3.3 General Magic의 Odyssey

Odyssey는 Telescript로 유명한 General Magic 사에서 개발한 이동 에이전트 시스템으로, Telescript의 개념과 기능이 최대한 Java 기반으로 이식되었다. General Magic사는 1993년에 다소 논쟁은 있지만 "mobile agent"라는 신조어를 만들고 1997년 US 특허를 받았다. 자사의 Telescript가 뛰어난 기술임을 인정받고도 급부상한 Java와의 경쟁에서 밀리게 되자 결국은 Telescript를 시장에서 공식적으로 철수한 후 내세운 제품이다.

Odyssey는 Odyssey 클래스 라이브러리(Odyssey class library)를 제공하고, 사용자는 이것을 이용하여 자신의 이동 에이전트 응용을 작성한다. Odyssey는 Java RMI를 확장하여 사용하기 때문에 JDK1.1이후의 버전에서 사용할 수 있고, 에이전트 전송(agent transport)을 위하여 CORBA의 IIOP(Internet Inter-ORB Protocol)와 Microsoft의 DCOM(Distributed Common Object Model)을 제공한다. 멀티에이전트 전송 메커니즘을 제공하는 유일한 시스템이다. 현재 Odyssey의 버전은 CORBA 객체와 JDBC를 통한 관계형 데이터베이스에 원격 접근(remote access)을 지원한다.

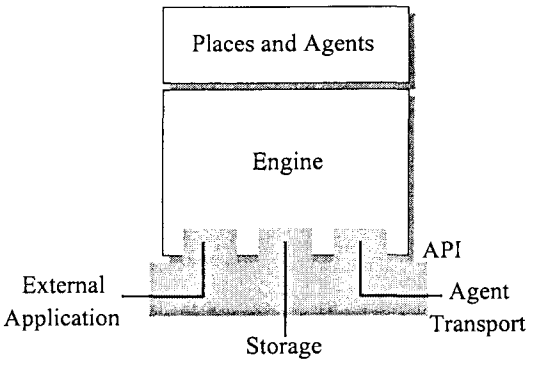

그림 7. Odyssey 에이전트 시스템의 개념적 구조

그림 7은 Odyssey의 개념적 구조를 보여준다[11]. Agent는 사람이나 조직을 위하여 자율적으로 행동하는 프로세스로서 작업을 수행할 수 있도록 쓰레드를 갖고있다. Place는 이동 에이전트 시스템의 수행 환경으로서 접근 기능(access control)등을 제공하고 이동 에이전트가 Place에서 Place로 이동하도록 한다. 이동 에이전트 시스템(엔진)은 에이전트의 생명 주기(생성, 관리, 해석, 전송, 폐기, 등)를 수행하는 플랫폼이다. 개념적으로 엔진은 적어도 3개의 API(Application Programming Interface)를 갖는다. Storage API는 디스크에 Place와 Agent를 저장하는데 사용되고, Transport API는 이동 에이전트를 전송할 때 통신하기 위함이다. external application API는 다른 언어로 작성된 응용과 상호 동작할 때 사용한다.

그림 8은 Odyssey 엔진의 개념적 구조를 보여주고 있다. 이동 에이전트는Place간을 이동할

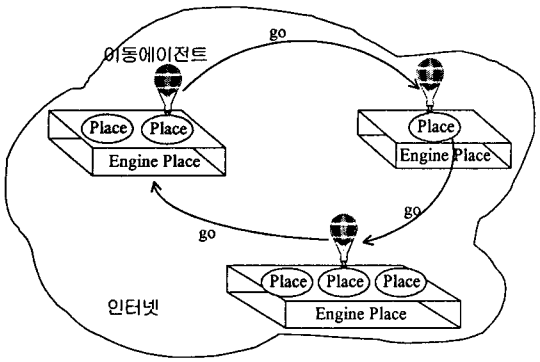

그림 8. Odyssey 엔진의 개념적 구조

수 있다. Place는 엔진 내부에 존재하는 논리적인 영역이다. 하나의 호스트에 하나의 엔진을 갖고 있고, 하나의 엔진은 하나의 engine place를 가지며 그 내부에 여러 개의 Place를 지원할 수 있다. engine place는 Region이라 불리는 더 큰 문맥(context) 내부에서 수행된다. Odyssey의 기본 이동 명령어는 go 명령어와 ticket 인자를 제공한다. ticket 인자는 다양한 형태로 목적지 호스트를 기술하는데 사용한다. 어떤 Place로 go 명령어가 수행되자마자 그 Place를 관리하는 엔진은 이동 에이전트를 인지하고, 접근을 허용할 것이지를 판단한다. 접근이 거절(deny)되면 이동 에이전트는 Purgatory라 불리는 특별 장소로 이동된다. Purgatory는 짧은 시간동안 제한된 환경에서 수행되도록 하는 곳으로서 그 이동 에이전트를 복구하고 새로운 목적지를 선택할 수 있도록 한다.

3.4 에이전트 시스템들의 비교 분석

앞에서 살펴본 에이전트 시스템들은 Java 기반 이동 에이전트 시스템들로 개발 플랫폼으로 Java를 사용하였기 때문에 공통적인 특성을 갖고 있다[5]. 첫째, 형태는 다르지만 이동 에이전트 서버(agent server)를 제공한다. 에이전트 서버는 어떤 머신의 접촉점(contact point)으로서 이동 에이전트가 이동하고 활동하는 영역을 제공한다. 둘째, 이동 에이전트는 서버간 이동할 수 있고, 이동할 때 상태(state)도 함께 이동된다. 이동할 때 순회(itinerary)에 따라 자동적으로 이동하거나 이동 에이전트 스스로 갈 곳(destiny)을 정하는 경우도 존재한다. 셋째, 이동 에이전트는 다양한 소스로부터 코드를 로드할 수 있다. 모든 이동 에이전트 시스템이 Java 클래스 로더를 사용하기 때문에 자기 파일 시스템, 웹, ftp 서버 등으로부터 Java 클래스 파일을 로드할 수 있다. 넷째, 이동 에이전트 시스템은 100% 순수 Java이고 JDK1.1의 특징을 사용한다. 이동 에이전트는 JDK1.1 호환 Java 가상 기계가 설치된 어떤 컴퓨터에서도 실행될 수 있다. 그러나, Java 구현 특성의 호환성 미비로 모든 이동 에이전트 시스템이 어떤 가상 기계에서라도 적절히 동작되지는 않는다.

에이전트 시스템들의 비교 분석은 표 1과 같이 IBM사의 Aglets Workbench, ObjectSpace사의 Voyager, General Magic 사의 Odyssey에 대해 여러 관점에서 정리해 보았다[12].

4. 이동 에이전트의 응용 분야

이동 에이전트가 가지는 일반적인 특징은 다음과 같다[13]. 첫째, 객체 전송(object passing)이다. 이동할 때 자체의 코드, 데이터, 방문 리스트 모두가 이동한다. 둘째, 자율적(autonomous)이다. 스스로 임무를 수행할 능력을 갖고있기 위해서 목적지에서 수행할 업무에 대한 정보를 가지고 이동한다. 셋째, 비동기적(asynchronous)이다. 자기 자신의 실행 쓰레드를 가지고 있으며 다른 에이전트와는 독립적으로 실행할 수 있다. 넷째, 지역 대화(local interaction)가 가능하다. 다른 이동 에이전트 및 호스트 내부에 있는 정적인 에이전트와 통신이 가능하고, 필요한 경우 통신을 위해 새

표 1. 에이전트 시스템들의 특징별 비교 분석

특징 \ 제품	IBM의 Aglets	ObjectSpace Voyager	General Magic의 Odyssey
Remote creation of agents	No	Yes	No
Remote java message	None	State-of-the-art	None
Messaging to mobile agents	None	Transparent	None
Messaging mode	Sync, Future	Oneway, Future, sync	None
Life spans	Explicit	5 different modes	Explicit
Mobile naming service	None	Integrated	None
Move to program	Yes	Yes	Yes
Move to object	No	Yes	No
Itineraries	Yes, special API	Yes, no special API	Yes, special API
Connectivity	Restricted	Full	Restricted
Security	Security manager	Security manager	None
Pricing	Free noncommercial	Free commercial	Free noncommercial
Size	120K without ORB	140K	230K
Future Direction	No stated direction	Rich feature set	No stated direction

로운 이동 에이전트를 생성, 파견할 수 있다. 다섯째, 병렬 수행(parallel execution)이다. 동시 다발적인 임무 수행을 위해 여러 개의 이동 에이전트를 여러 장소로 보내서 실행시킬 수 있다.

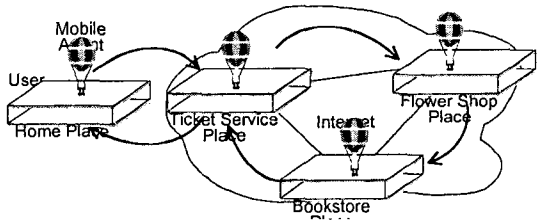

그림 9. 이동 컴퓨팅 환경에서 분산 응용 모델링

이러한 에이전트 기술을 이용하면 그림 9에서와 같이 분산 응용 서비스는 에이전트의 집합으로 모델링할 수 있고, 각 에이전트는 생존의 터전인 플레이스(place)를 사용한다. 특히, 인터넷은 광활한 정보의 보고로서 플레이스의 집합으로 모델링할 수 있다. 뿐만 아니라 에이전트 자체가 이동하기 때문에 보다 확장성(scalable) 있는 응용을 작성할 수 있다.

이동 에이전트는 실행 코드(executable code)가 목표 달성을 위해 호스트를 이동하며 작업을 수행한다. 이러한 이동 에이전트 패러다임에 적합한 응용은 첫째, 전자상거래(electronic commerce)이다. 상용 트랜잭션은 주식시세(stock quote) 접근과 같이 원격 자원을 실시간 접근(real-time access)하거나 에이전트간 협상을 요구한다. 각 에이전트는 자신의 목적(goal)을 갖고, 이를 성취하기 위하여 자신의 전략에 따라 동작하도록 프로그래밍 되어야 한다. 둘째, 분산 정보 검색(distributed information retrieval)이다. 검색 엔진(search engine)은 검색 목록(search index)을 작성하기 위하여 많은 양의 데이터를 이동시키지만, 이동 에이전트는 정보가 있는 곳으로 이동 에이전트를 급송(dispatch)하여 그 곳에서 검색 목록을 작성한 후 원래의 위치로 되돌아온다. 셋째, 정보

산포(information dissemiantion)이다. 흔히 말하는 인터넷의 푸시(push) 기술을 쉽게 구현할 수 있다. 뉴스나 소프웨어 업그레이드(software upgrade)와 같은 정보를 산포하고자 할 때 이동 에이전트는 새로운 소프트웨어 및 설치 절차 등을 고객의 컴퓨터로 가져와서 자동으로 갱신하고 고객 컴퓨터의 소프트웨어를 관리할 수 있다.

이밖에 통신 네트워크 서비스(telecommunication networks service), 개인 비서(personal assistance), 보안 중개(secure brokering), 워크플로우와 그룹웨어(workflow applications and groupware), 병렬처리(information processing), 모니터링과 통지(monitoring and notification) 등의 응용에 적합하다[2].

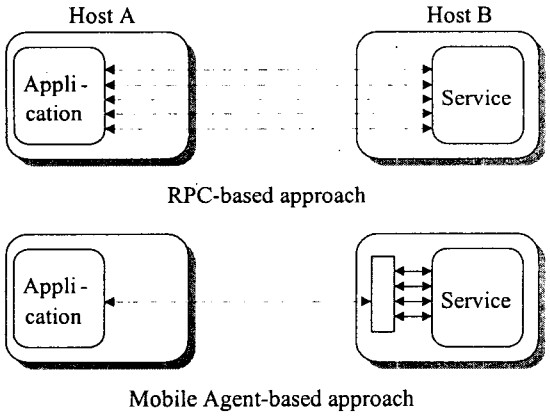

그림 10. 이동 에이전트의 네트워크 트래픽 감소

이동 에이전트는 그림 10에서 보는 바와 같이 RPC 기법보다 네트워크 트래픽을 감소시킨다. 이동 에이전트는 통신할 많은 대화를 한 묶음으로 묶어 그것을 목적지 호스트로 전송한다. 목적지에 도착하면 그 시스템 내에서 국부적으로 상호대화를 수행하고 결과를 되돌려 보낸다. 따라서 저속 회선에서의 대용량 대화(high-bandwidth conversation) 응용에 적합하다. 이동 에이전트는 네트워크 연결을 계속 유지한 상태로 작업하지 않고 실행 코드 자체가 시스템간을 이동하며 임무를 수행하기 때문에 네트워크 연결이 불안정하거나 부하가 많이 걸리는 환경에서 유용하게 활용될 수 있다[14]. 특히, 이동 컴퓨팅 환경은 부분적으로 연결된 컴퓨터(partially connected computer)와 공동 작업하는 경우가 많다. 왜냐하면 이동 호스트가 사용자의 의지에 따라 네트워크 연결을 끊거나, 무선 매체의 불안정 상태로 인한 네트워크 연결이 끊어지는 경우가 존재한다. 자발적으로 네트워크 연결을 절단한 경우에는 밧데리를 절약하기 위하여 전원을 끈 경우, 통신 요금을 절약하기 위하여 네트워크 연결을 절단한 경우 등이다.

5. 결론 및 향후 연구 방향

대등 관계 응용 모델에 따른 이동 에이전트 시스템은 다음과 같은 장점을 갖는다. 첫째, Jump-do-Jump 형태의 분산 응용 프로그래밍에 적합하다. 둘째, 클라이언트-서버 모델에서는 서버가 정의한 고정 서비스만 사용할 수 있지만 대등 관계 형태의 이동 에이전트는 수행 능력과 어느 정도의 지능이 있는 코드가 직접 가서 수행하기 때문에 융통성을 높여준다. 셋째, 전체적인 시스템 부하의 균형을 촉진할 수 있다. 이동 에이전트가 시스템 부하를 고려하여 라우팅 한다면 시스템 부하가 적은 곳에서 수행할 수 있기 때문이다. 넷째, 망 대역폭을 효율적으로 이용할 수 있다. 예를 들어, 기상 정보 데이터를 메시지 형태로 가져와서 분석하는 것보다 에이전트가 데이터 있는 곳으로 직접 가서 분석한 다음 그 결과만 본다면, 망 대역폭을 훨씬 절약할 수 있기 때문이다. 다섯째, 분산 응용이 이동 컴퓨팅 환경에서도 효율적으로 수행할 수 있는 환경을 제공한다. 이동 호스트에서 이동 에이전트를 보내놓고 이동 호스트의 전원을 끈

다음 결과를 보고자 할 경우 다시 연결하여 볼 수 있기 때문이다.

앞으로의 연구 방향은 이동 에이전트 기술이 정보 산업에 새로운 활력을 불어넣을 신기술이 되도록 하는 측면의 노력이다. 앞의 이동 에이전트 시스템들의 비교 분석에서 보았듯이 기술과 산업이 새롭기 때문에 구조나 구현 측면에서 서로 다른 점이 많다. 이러한 점은 이동 에이전트 기술들의 상호운용성(interoperability)이나 신속한 기술 발전 및 산업 성장에 많은 저해 요소가 되고 있다. 이러한 문제점들을 효율적으로 해결하기 위해서는 이동 에이전트 기술의 표준화가 시급한 실정이다.

참 고 문 헌

[1] C. G. Harrison, D. M. Chess, and A. Kershenbaum, "Mobile Agents: Are they a good idea?," Technical Report, IBM T. J. Watson Research Center, p.21, Mar. 1995.

[2] D. B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing?," Proceedings of European Conf. on Object Oriented Programming'98, Jan. 1998.

[3] P. E. Renaud, Intro. To Client/Server Systems: Practical Guide for Systems Professionals, Wiley & Sons, 1993.

[4] S. Versteeg and L. Sterling, "Languages for Mobile Agents," http://www.cs.mu.oz.au/~scv/thesis.html, Aug. 1997.

[5] J. Kiniry and D. Zimmerman, "A Hands-on Look at Java Mobile Agents," IEEE Internet Computing, Vol. 1, No. 4, Jul./Aug. 1997.

[6] Agent Society, "Agent Standards Activities", http://www.agent.org/pub/standards.html.

[7] D. B. Lange, "Mobile Agents with Java: The Aglet API," World Wide Web Journal, 1998.

[8] ObjectSpace, "Voyager : Core Package Tecnical Overview,"http://www.objectspace.com/Voyager/voyager.html, 1997.

[9] ObjectSpace, "Voyager : ORB3.0 Developer Guide," ObjectSpace Inc. http://www.objectspace.com/Voyager/, 1999.

[10] G. Glass, "The ObjectSpace Voyager Universal ORB," ObjectSpace Inc. http://www.objectspace.com/Voyager/, 1999.

[11] J. White, "Mobile Agents White Paper," General Magic White paper, http://www.genmagic.com/agents/Whitepaper/whitepaper.html, p.28, 1996.

[12] ObjectSpace, "A Comparison : ObjectSpace Voyager, General Magic Odyssey, IBM Aglets," http://www.objectspace.com/Voyager/, 1997.

[13] A. Chavez & P. Maes, "Kasbah:An Agent Marketplace for Buying and Selling Goods," Proc. Of 1st Int l Conf. On the Practical Application of Intelligent Agents & Multi Agent Technolog, 1996.

[14] R. Gray, D. Kotz, S. Nog, D. Rus and G. Cybenco, "Mobile Agents for Mobile Computing," Proc. of the 2nd Aizu Intl Symposium on Parallel Algorithms/Architectures Synthesis (pAs97), Fukushima, Japan, p.17, Mar. 1997.

김 평 중

- 1985년 충남대학교 계산통계학과 전산학 전공 (이학사)
- 1995년 KAIST 전산학과 (공학석사)
- 1995년 정보처리기술사 취득
- 1987년~1988년 포항종합제철㈜ 전산시스템부 (전산기술직)
- 1988년~1998년 한국전자통신연구원 멀티미디어연구부 (선임연구원)
- 1998년~현재 충북도립옥천대학 컴퓨터정보과(전자계산소장)
- 1998년~현재 한국정보처리학회 학회지 편집위원
- 관심분야 : 이동에이전트, 전자상거래, 컴퓨터네트웍, 분산 멀티미디어

진 성 일

- 1978년 서울대학교 계산통계학과 전산학 전공 (이학사)
- 1980년 KAIST 전산학과 (공학석사)
- 1994년 KAIST 전산학과 (공학박사)
- 1987년~1989년 Northwestern대학 전산학과 객원교수
- 1990년~1992년 충남대학교 전자계산소 소장
- 1983년~현재 충남대학교 컴퓨터과학과 정교수
- 1989년~현재 한국정보과학회 데이터베이스연구회 운영, 편집,협력위원
- 1995년~현재 한국정보처리학회 멀티미디어연구회 운영위원
- 1996년~현재 데이터베이스학회 편집위원
- 1996년~현재 충남대학교 소프트웨어 연구센터 소장
- 1996년~현재 한국정보과학회 평의원
- 1996년~현재 CALS/EC 학회 이사 겸 S/W전문분과 위원장
- 1997년~현재 대전 소프트웨어지원센터 운영위원장
- 관심분야 : 정보모델링, 데이타베이스, CALS/EC, 멀티미디어, 성능분석