이동형 에이전트와 Java 기반의 Aglets 구현 기술

임준식* · 황대훈*

1. 서 론

오늘날 전세계 곳곳의 수많은 유용한 정보들이 인터넷이라고 하는 거대한 정보 네트웍으로 연결되고, 이를 이용하고자 하는 사용자의 수요 요구가 폭발적으로 증가하는 추세에 있다. 그러나 인터넷의 정보 홍수 속에서 사용자가 원하는 정보만를 정확하고 신속하게 제시간에 얻기란 쉬운 일이 아니며, 따라서 사용자들은 자신이 필요로 하는 정보만을 손쉽게 접할 수 있도록 지원하는 기능을 요구하게 되었다.

이에 따라 90년대 중반에 들어서면서 이러한 작업을 대신해주는 에이전트에 대한 관심과 필요성이 대두되었고, 이와 함께 에이전트의 중요성 및 역할 또한 점점 커지고 있다. 또한 국가의 정책 차원에서 추진되고 있는 초고속 정보통신망과 연계하여 온라인 쇼핑 등의 전자상거래나 메시지 처리 시스템 등과 같은 이동형 컴퓨팅(mobile computing) 분야에서도 에이전트의 중요성 및 필요성이 대두되고 있다.

에이전트(agent)란 용어에 대한 정의를 한마디로 표현하기는 용이하지 않으며, 에이전트를 보는 시각에 따라 다양하게 정의되고 있다. 그러나 이에 대한 대표적인 정의를 살펴보면, '사용자가 원하는 작업을 사용자를 대신하여 해결하여 주는 소프트웨어' 또는 '특정 목적을 수행하기 위하여 사용자를 대신하여 작업을 수행하는 자율적인 프로세스(autonomous process)'라고 할 수 있다 [2,8].

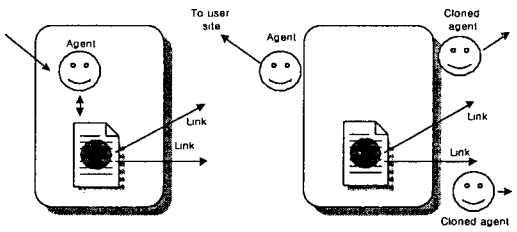

그림 1. 이동형 에이전트가 HTML 문서를 액세스하기 위하여 도착한 후, 또 다른 문서로의 링크를 위하여 새로운 에이전트를 생성하는 과정[5]

위의 정의에서와 같이, 에이전트는 주어진 작업만을 수동적으로 수행하기보다는 목표지향성(goal-directness)을 가지고 자신의 목적 달성을 스스로 추구하는 능동적인 자세를 가진다. 또한, 에이전트는 그림 1과 같이 자율성(autonomy)을 가지고 사용자나 조직의 권한을 대행하여 특정 기간 동안 혼자 독립적으로 수행될 수도 있고, 또는 사회성(social ability)을 가지고 다른 에이전트와의 상호 교류를 통하여 보다 복합적인 기능을 수행할 수도 있다. 에이전트는 이외에도 반응성(reactivity), 이동성(mobility) 및 이성(rationality) 등과 같은 속성을 가지는 자율적인 프로그램이다.

본고에서는 이동형 에이전트와 이의 구현을 위하여 가장 널리 사용되는 언어인 Java를 정의하

*경원대학교 전자계산학과 교수

고, 4장에서는 이의 대표적인 시스템인 Aglets을 소개한다. 5장은 Aglets 구현을 위한 기본 프로그램 예제를 중심으로 설명한다.

2. 이동형 에이전트

2.1 분류에 따른 에이전트

에이전트는 그 적용 분야나 목적에 따라 여러 가지 형태가 있을 수 있으나, 이들 여러 종류의 에이전.트들을 관점에 따라 분류하면 다음과 같다 [9].

첫째, 기능적 관점으로서, 에이전트가 수행하는 기능이나 역할에 따라 인터페이스 에이전트(interface agent 또는 user agent)와 소프트웨어 에이전트(software agent 또는 task agent)로 분류한다.

둘째, 특성적 관점으로서, 에이전트의 여러 특성 중 특히 어떠한 성질에 주목하여 에이전트를 설계하는가에 따라 인텔리전트 에이전트(intelligent agent)와 협조 에이전트(cooperative agent)로 분류한다.

셋째, 구조적 관점으로서, 에이전트의 업무 수행을 위한 지식 보유 여부에 따라 지식형 에이전트(knowledge-based agent, cognitive agent), 반사형 에이전트(reflexive agent 또는 reactive agent) 및 혼합형 에이전트(hybrid agent)로 분류하기도 한다.

이외에 에이전트의 이동성 관점에 따라, 고정형 에이전트와 이동형 에이전트로 구분할 수 있다 [8,11].

고정형 에이전트(stationary agent)는 그 실행이 시작된 자체 시스템에서만 실행된다. 따라서 만약 자신이 필요한 정보가 자체 시스템에 없거나 다른 시스템의 에이전트와 상호 작용을 할 필요가 있을 시에는, RPC(Remote Procedure Call) 등과 같은 별도의 통신 방법을 이용하여야 한다.

그러나 이동형 에이전트(mobile agent)는 그 실행이 시작된 시스템에 구애받지 않고, 필요하다면 네트웍 상의 다른 호스트들 간을 옮겨다니면서 실행될 수 있다. 즉, 고정형 에이전트와 비교해 보면 서로 다른 시간에 서로 다른 시스템에서 수행될 수 있는 것이다. 이와 같이 이동형 에이전트는 이질적인 망(heterogeneous network)에서 스스로의 제어로 호스트 사이를 이주(migration)하고, 각 호스트의 다른 에이전트와 상호 동작하거나 자원을 이용하여 부여된 임무를 수행하고, 수행이 끝났을 경우에는 최초에 시작된 호스트(home)로 되돌아온다.

2.2 이동형 에이전트의 필요성

이와 같은 이동형 에이전트가 오늘날 부각되는 이유 중에는, 컴퓨터 기술과 통신 기술의 자연스러운 상호 결합을 통한 새로운 컴퓨팅 서비스인 이동형 컴퓨팅의 등장을 들 수 있다. 이동형 컴퓨팅의 예로는 PDA(Personal Digital Assistant)나 노트-북 컴퓨터 등과 같은 이동형 호스트(mobile host)를 들 수 있는데, 이동형 호스트는 때와 장소에 구분없이 네트웍에 접속하여 각종 작업을 할 수 있다는 장점 때문에 오늘날 점차 널리 이용되고 있다. 그러면 이러한 이동형 호스트와 이동형 에이전트는 어떤 관련을 가지는 지를 살펴보자.

PDA나 노트-북 컴퓨터는 데스크-탑 컴퓨터에 비하여 연산 능력이나 입출력 기능이 떨어지는 단점을 가진다. 그럼에도 불구하고 PDA나 노트-북 컴퓨터를 이용하는 이유는 이동과 휴대가 용이하다는 장점 때문인데, 사용자가 이들 이동형 호스트를 이용하여 이동간에 복잡한 연산이나 사용자 인터페이스를 필요로 하는 작업을 수행하고자

하거나 인터넷에 접속하여 긴 시간에 걸쳐 여러 사이트를 내비게이션 하면서 필요한 정보를 검색하고자 한다면, 이는 분명 비효율적이고 불편한 일이 될 것임에 틀림이 없다. 바로 이러한 경우에 이동형 에이전트는 그 진가를 발휘할 수 있다.

이동형 호스트를 이용하는 사용자는 긴 시간을 요하는 작업이나 여러 과정을 요하는 검색을 수행하고자 하면, 이들 작업을 대신 처리하도록 에이전트에게 위탁하여 연산 속도가 보다 빠른 또는 현재 작업 부하가 많지 않은 네트워 상의 다른 컴퓨팅 호스트로 전송한다. 그후 이동형 호스트의 사용자는 또 다른 작업을 수행하거나 또는 전원을 끈 채로 대기할 수 있으며, 이동형 에이전트는 다른 호스트에서 실행이 계속될 것이다. 그후 이동형 에이전트에 부여된 임무가 끝나면 이동형 호스트로 그 결과만을 되돌릴 것이며, 이동형 호스트의 사용자는 자신이 필요한 시간에 맞추어 그 결과를 확인할 수 있다.

이로 인하여 사용자는 보다 빠른 시스템 응답 시간(response time)을 기대할 수 있고, 그림 2와 같이 네트워의 통신 부하(load)를 줄일 수 있을 뿐 아니라 네트워의 지연 현상(latency)을 극복할 수 있다[3,4].

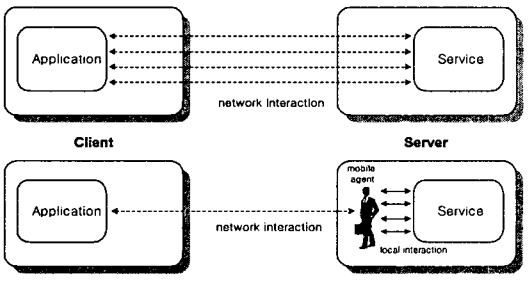

그림 2. Client-server와 이동형 에이전트의 통신 부하 비교

이와 같이 이동형 에이전트를 이용한 이러한 서비스 형태는 네트워 상의 인터넷 자원을 효율적으로 이용하는 분산 처리 서비스의 한 유형으로 분류될 수 있는데, 특히 이동형 호스트를 이용하는 경우 아주 효율적인 수행 환경을 제공한다. 이동형 에이전트 시스템은 분산 처리 시스템을 위한 응용 프로그램을 작성함에 있어 매우 적합한 형태로서, 에이전트가 언제, 어디로 이동한다고 기술되어 있으면 시스템이 알아서 전송하기 때문에 분산 응용 프로그램을 작성하는데 있어 편리하고 효율적일 뿐만 아니라 강력한 프로그래밍 형태를 제공한다.

2.3 이동형 에이전트 시스템의 구성 요소

이동형 에이전트 시스템의 개념적인 모델은 에이전트와 플레이스(place)라고 하는 두가지 핵심 개념에 기초를 둔다. 이 개념을 기반으로 에이전트의 행위(behavior)와 네트워 상의 이동 및 에이전트간 통신이 이루어 진다[8].

(1) 에이전트

에이전트는 상태, 실행, 인터페이스, 식별자 및 주체의 5가지 속성을 가지는 개체이다. 이러한 속성은 에이전트가 네트워을 이동할 때 가지고 다니는 것이다.

상태(state)는 에이전트의 이동 전에 수행하던 연산을 이동 후에도 계속(resume)하고자 할 때 필요한 속성이며, 실행(implement- ation)은 에이전트가 실행 장소에 구애받지 않고 실행하는데 필요한 실행 코드(implementation code)를 가지고 목적 호스트(destination host)로 갈 것인지에 대한 여부를 결정하는데 필요하다. 인터페이스(interface)는 다른 에이전트와의 통신을 위한 속성이며, 식별자(identifier)는 에이전트를 인식하고 이동중의 에이전트가 위치하는 곳을 파악하기

위한 것이다. 끝으로 주체(principal)는 에이전트를 생성하고 그의 행위를 책임지는 사람 또는 조직의 소재을 나타낸다.

(2) 플레이스

플레이스는 에이전트가 속해 있는 특정 위치에 관계없이 실행될 수 있는 환경을 제공한다. 따라서 플레이스를 에이전트를 위한 운영체제라고도 할 수 있는데, 엔진, 자원, 위치 및 주체 등 4가지 개념이 플레이스에서 중요한 역할을 한다.

플레이스는 독자적으로 에이전트를 실행할 수 없기 때문에, 에이전트와 플레이스가 실행될 가상 기계(virtual machine)의 역할을 수행하는 엔진(engine)이 있어야 한다. 다시 말해서, Java 기반의 이동형 에이전트 시스템에서 엔진을 Java 가상 기계 및 운영체제와 동일한 것으로 보아도 무방하다. 자원(resource)은 에이전트의 실행을 위하여 엔진과 플레이스가 제공하는 네트웍, 데이터베이스, 프로세서 및 소프트웨어 서비스 등의 시스템 자원을 통칭한다. 위치(location)는 이동형 에이전트에 있어 중요한 개념인데, 에이전트가 실행되는 위치를 정의한다. 주체(principal)는 플레이스가 수행하는 내용에 대한 책임을 가지는 주체를 나타낸다.

(3) 행위

하나의 에이전트는 플레이스에서 생성(creation)되고 소멸(disposal)되며 또한 플레이스 간을 이동(transfer)한다. 보다 자세한 내용은 4장에서 소개한다.

(4) 통신

에이전트는 같은 플레이스 내(intra-place)의 다른 에이전트와 통신 및 다른 플레이스(inter-place 및 inter-engine)에 있는 에이전트와의 통신을 위하여 1대1(peer-to-peer)[5] 또는 전역전송(broadcasting) 방식으로 메시지를 전송할 수 있다.

3. 에이전트 프로그래밍 언어

3.1 분류

C나 Pascal 또는 Prolog 등과 같은 기존의 프로그래밍 언어를 이용하여 위와 같은 이동형 에이전트 시스템을 구현하는 것이 불가능한 것은 아니나, 매우 비효율적인 것만은 틀림이 없다[2]. 그 이유는 기존의 프로그래밍 언어로 작성된 프로그램은 실행 속도를 최소화하기 위한 기계어 코드 생성에 초점을 맞추어 번역되기 때문에 플랫폼에 매우 의존적이다. 또한 번역된 프로그램 코드들은 한 호스트내의 메모리 영역(address space)에서 실행되기 때문에, 다른 호스트로의 코드 이동 및 실행을 위해서는 프로그래머가 이를 별도의 프로그램으로 관리하여야 한다.

이동형 에이전트는 네트웍을 이동하며 수행되기 때문에 프로그램 코드뿐만 아니라 상태 정보를 포함한 각종 속성 정보도 이동되어야 한다. 이때 코드는 이동되는 모든 호스트에 같은 형식으로 실행되어야 하고, 다시 컴파일(re-compile)하지 않고도 실행될 수 있거나 바로 인터프리팅(interpreting)될 수 있는 이식성 있는 중간 형태의 언어(potable intermediate interpreter-based language) 형태를 가져야 한다. 상태 정보는 이동 코드의 수행 위치, 중간 결과 등을 계속 유지해야 하기 때문에 지속성(persistency)이 있어야 한다. 또한 이동 전에 수행되었던 곳에서부터 다시 수행되도록 하기 위해서는 지속성 상태를 코드 부분에 표현하거나 부호화된 스택(encoded stack)으로 전송하여야 한다.

따라서 에이전트 언어는 이와 같은 요구 사항을 충족하면서도 2.3절에서 소개된 여러 개념을

표 1. 에이전트-기반의 응용 프로그램 개발을 위한 언어의 분류[2]

Agent type	Language Class	Examples
Collaborative agent	Actor Language	Actors
	Agent-oriented programming language	Agent-0 Placa
Interface agent Information agent Mobile agent	Scripting language	TCL/Tk
		Java
		Telescript Active Web tools
		Python
		Obliq
		April
		Scheme-48
Reactive agent	Reactive language	RTA/ABLE

효율적으로 프로그래밍 할 수 있어야 한다. 에이전트 언어는 크게 에이전트 기술 언어와 에이전트 개발 언어로 구분된다[17].

에이전트 기술 언어(agent specification language)는 에이전트의 속성과 행동 등을 기술하기 위한 언어로써, 주로 에이전트가 수행할 수 있는 기능, 목적, 계획, 지식, 믿음 스키마(belief schema) 등의 형태로 에이전트를 표시하게 된다. 대표적인 에이전트 기술 언어에는 AOP[18]와 이의 향상된 버전인 PLACA 등이 있다.

에이전트 개발 언어(agent programming language)는 실행 가능한 에이전트를 프로그래밍하기 위한 언어로써 대부분 해석 언어(interpreter language)의 형태를 가지며, 객체 지향 기법을 사용함으로써 플랫폼에 독립적인 에이전트의 개발을 가능하게 한다. 대표적인 에이전트 개발 언어로는 Java, TCL /Tk(Tool Command Language/ Toolkit)[7], Telescript 등이 있다. TCL/Tk는 비교적 간단한 스크립트 언어로써, 여러 플랫폼에서 가용성(availability)을 가지며 GUI의 생성을 지원하는 API로서 Tk를 지원한다. 이들 언어의 공통적인 특징은 스크립트 언어와 매우 유사한 구조를 가지며, 에이전트 프로그램이 컴파일되지 않고 목적지 호스트에서 인터프리팅 된다는 점이다.

이중에서도 Java와 Telescript[4]는 객체지향 언어이면서 인터프리터 언어로서, 가상 기계(virtual machine) 개념을 도입하여 특정 플랫폼에 독립적인 에이전트를 생성하는 것이 가능하다는 공통점을 가진다. 차이점으로는 Java에서는 클라이언트의 요청에 의하여 서버의 프로그램이 클라이언트로 이동하여 수행되는 반면, Telescript는 반대로 클라이언트의 에이전트를 구성하는 해당 코드가 서버로 이동하여 수행된다는 점이다. 이로 미루어 어느 것이 이동형 에이전트 언어로서 더 적합하다기 보다는 두 언어가 상호 보완적인 관계에 있다고 할 수 있다.

이중 Java는 "better C++"라고 불릴 만큼 C++과 유사한 객체지향 언어이며, 인터넷 언어로서 네트웍 환경에 유용한 언어이다. 특히 다음과 같은 장점으로 인하여, Java는 이동형 에이전트의 프로그래밍에 매우 적합한 언어로 인식되고 있다[2,8].

첫째, Java는 이기종 환경에서 수행되도록 설계되었기 때문에, Java 컴파일러가 생성한 Java 응용 프로그램의 바이트 코드(byte code)는 네트웍상의 어떤 호스트에서도 기종에 관계없이(platform-independent) 실행될 수 있다.

둘째, Java는 인터넷과 인트라넷에서의 사용을 목표로 개발되었기 때문에, 보안(security)에 대한 요구를 설계에 다양한 방법으로 반영하였다.

셋째, Java 가상 기계는 실행 시에 클래스를 로딩하고 정의하는 동적 클래스 로딩(dynamic class

loading) 메커니즘을 허용한다.

넷째, Java는 멀티스레드 프로그래밍과 동기화 프리미티브를 언어 자체에서 지원한다. 이를 통하여 자신 고유의 스레드(thread)로 실행하는 각 에이전트가 자율적으로 행동할 수 있다.

이외에도 Java 언어는 객체 직렬화(object serialization)와 reflection 등을 장점으로 가진다. 반면, Java 언어를 이용하여 이동형 에이전트를 구현하는 것은 Java 언어의 시스템 자원에 대한 부적절한 제어 능력 등 몇가지 단점도 있다.

3.2 이동형 에이전트 시스템

이동형 에이전트 시스템은 이동형 에이전트를 생성, 이동, 수행, 전송, 해석 및 폐기 등 에이전트의 생명 주기를 관리할 수 있는 플랫폼이다. 이동형 에이전트 시스템은 에이전트를 프로그래밍하기 위한 언어, 에이전트를 설치하기 위한 라이브러리, 에이전트를 수행시키기 위한 인터프리터, 에이전트를 전송하기 위한 프로토콜 및 에이전트를 보호하기 위한 보안 등의 측면으로 분류될 수 있다[11].

본고에서는 프로그래밍 언어에 따르는 분류로서, 특히 Java에 기반한 시스템과 그에 기반하지 않은 시스템으로 분류하여 설명하고자 한다.

Java에 기반하지 않은 이동형 에이전트 시스템은 다음과 같다. 독일의 Kaiserslautern 대학에서는 Tcl/Tk를 이용하여 UNIX 시스템에 구현하는 이동형 에이전트 시스템인 Ara(Agent of Remote Action)[15]의 개발을 시작하였다. 노르웨이의 Tromso 대학과 미국의 Cornell 대학이 공동으로 개발하는 TACOMA(Tromso And Cornell Moving Agents)[1]는 UNIX 시스템과 TCP(Transmission Control Protocol)에 기초하여 C, TCL/Tk, Perl 등과 같은 언어로 작성된 에이전트를 지원한다.

표 2. Java에 기반한 이동형 에이전트 시스템의 분류

Java-based Mobile Agent System	Non-Java Mobile Agent System
Odyssey (General Magic)	ARA (University of Kaiserlautern)
MOA (OSF)	TACOMA (Univ. of Tromso & Cornel Univ.)
Mole (Univ. of Stuttgart)	Agent Tcl (Dartmouth Collegd)
Concordia (Mitsubishi)	Telescript (General Magic)
Aglets (IBM Tokyo R. Lab)	Obliq (DEC)

이외에도 Dartmouth 대학에서 정보 검색을 효율적으로 수행하도록 Tcl/Tk를 이용하여 개발한 Agent Tcl[1,12] 등이 있다[7].

Java에 기반한 이동형 에이전트 시스템으로 대표적인 것은 Odyssey이다. Odyssey[1]는 General Magic사의 상용 제품으로써, 자사의 에이전트 개발 언어인 Telescript 개발 기술을 Java에 접목하여 개발한 이동형 에이전트 시스템이다. Odyssey는 Odyssey 클래스 라이브러리(Odyssey class library)를 제공하고, 사용자는 이를 이용하여 자신의 이동형 에이전트를 개발할 수 있도록 지원하고 있다.

MOA(Mobile Objects and Agents)[14]는 OSF RI(Open Software Foundation Research Institute)에서 수행 중인 프로젝트이다. 여기에서 이동 객체(mobile object)는 thread를 가지고 능동적으로 동작하고 상태 정보를 유지하면서 지능적인 결정을 하는데, 이 이동 객체가 사용자 측면에서 동작할 때 에이전트라고 한다.

이밖에도 Java에 근거한 이동형 에이전트 시스템으로는, 독일 Stuttgart 대학의 Mole[1,16]이나 일본 Mitsubishi Electronics사 정보기술센터의 Concordia 등이 있는데, 전반적인 경향은 Java 언어에 근거한 기반 구조(infrastructure)를 개발하

는데 초점을 맞추고 있다.

특히 본고에서는 Aglets에 초점을 맞추어 설명하고자 하는데, Aglets Workbench [1,8,13]는 IBM Tokyo Research Lab에서 개발한 것이다. 보다 자세한 내용은 다음 장에서 소개한다.

4. Aglets

Aglets[4,8]은 이동형 에이전트 모델의 하나로서, Java에 근거한 자율적인 이동형 에이전트 시스템(automous mobile agent system)이다. Aglets은 aglet이라고 하는 Java 객체를 기반으로 하는데, aglet은 agent와 Java applet의 합성어로서 이동성과 영속성 및 그 자신의 실행 스레드를 가지고 네트워을 이동하면서 다른 aglet과 통신을 수행할 수 있는 복합 Java 객체이다. Aglets은 Java applet과 이벤트 위임처리 모델(event delegation model)을 참조하여 설계되었기 때문에, Java 프로그래머들은 Aglets을 쉽게 배울 수 있으며 이동형 에이전트의 장점을 쉽게 이용할 수 있다. 또한 Aglets을 이용하여 프로그래머는 분산 환경에서 쉽게 자율 객체를 구현할 수 있다.

최초의 Java 기반의 이동형 에이전트 시스템인 Aglets의 알파 버전 ASDK(Aglets Software Development Kit)는 1996년에 발표되었는데, 이는 Aglet API 패키지, 도큐먼트, 샘플 aglet 및 aglet 서버 등으로 구성되어 있다. 여기에서 aglet 서버는 aglet이 실행될 수 있는 환경을 제공하는데, aglet을 관리하고 원격 호스트로 aglet을 전송하고 수신하는 기능을 수행한다. Aglet API는 사용자가 이동형 Java 에이전트를 구현하도록 Java 클래스와 인터페이스를 지원하는 에이전트 개발 킷트이다.

Aglets 객체 모델은 인터넷과 같은 개방형 광역 네트워에서 이동형 에이전트 기술을 증폭시키는데 필요한 개념(abstraction)과 행위(behavior)를 정의하는 것이다.

Aglets 모델은 aglet, 프록시, context, 식별자(identifier) 등을 주된 개념으로 포함하며, 이 개념들이 aglet이 임무를 수행할 수 있는 환경을 제공한다[1,8].

Aglet은 네트워 상의 aglet이 실행 가능한 호스트를 방문하여 그곳에서 실행되는 이동형 Java 객체이다. 프록시(proxy)는 aglet의 대리인 역할을 하여 aglet이 공용 메소드(public method)에 의해 직접 액세스되지 않도록 보호하며, aglet의 실제 위치가 노출되지 않도록 aglet 위치에 대한 투명성을 제공한다. context는 aglet의 작업 공간으로서, 2.3절에서 소개한 플레이스에 상응한다. 이는 aglet이 동일한 실행 환경에서 실행될 수 있도록 이를 관리하는 고정형 에이전트이다. 식별자는 각 aglet에 유일하게 부여되며, 종료 시까지 불변한다.

Aglet에 의해 수행되는 행위는 이동형 에이전트의 속성에 대한 세밀한 분석을 통하여 6가지로 정의되는데, 보다 자세한 내용은 다음 장에서 소개한다.

5. 이동형 에이전트 프로그래밍

이번 장에서는 Aglets 모델에서 제공하는 이동형 에이전트의 이동성과 그 프로그래밍 대해서 서술한다. Aglets 모델은 Java가 가지고 있는 에이전트의 특징을 수용하면서 Java의 단점을 보완하고 있으며, 인터넷이나 개방형 WAN(open wide-area networks) 등에서 이동형 에이전트에 필요한 aglet, proxy, context 등과 같은 개념(abstraction)들을 가지고 있다. 이 Aglets 모델은 alget이라는 Java 객체를 호스트간에 이동시킬 수 있도록 구성되어 있다. 이러한 이동성을 위한

aglet의 실행은 다음과 같은 기본 연산들에 의해 이루어진다.

- Creation: Creation은 context라고 불리는 서버의 주소와 이름으로 구성된 작업실에서 발생한다. 이 context에서 Java의 이동 객체인 aglet이 생성되면서 식별자가 주어지고 초기화 된다. 초기화가 되면 aglet은 곧 실행을 시작한다.
- Cloning: aglet의 cloning은 같은 context 안에서 원래의 aglet과 똑같은 aglet을 복사하는 것이다. 그러나 이 복사된 aglet은 자신의 식별자를 가지며 실행을 다시 시켜야 한다.
- Dispatching: dispatching이란 현재의 context에서 한 aglet을 제거하고, 이동될 context 안에 그 aglet을 추가하여 실행시킨다는 의미이다.
- Retraction: retraction이란 한 aglet이 현재의 context에서 제거되고 그 aglet을 요청한 context 안에 추가된다는 의미로 dispatching에 비해 수동적이라는 점이 다르다.
- Activation과 Deactivation: deactivation은 한 aglet의 실행을 임시로 중지한 후 그의 상태를 디스크에 저장하는 것을 말한다. activation은 이러한 aglet을 같은 context 안에서 다시 실행함을 말한다.
- Disposal: aglet의 disposal은 현재 실행을 중지시키고 그 aglet이 있는 context로부터 제거하는 것이다.

다음 그림 3은 이러한 과정에 대한 aglet의 life-cycle 모델을 표현한 것이며, 그림 4는 이들 중 한 aglet이 dispatching되는 과정을 보여주고 있다. 그림에서 Byte Code와 State로 표현된 aglet은 Host A로부터 Host B로 전달되고 있다. 여기서 State 정보는 에이전트가 이동을 한 후에 다시 실행하기 위한 정보를 갖고 Code는 그 실행 코드를 의미한다. 실제로는 그 aglet은 Host A에서 없어지고 똑같은 aglet이 Host B에 생성 된다.

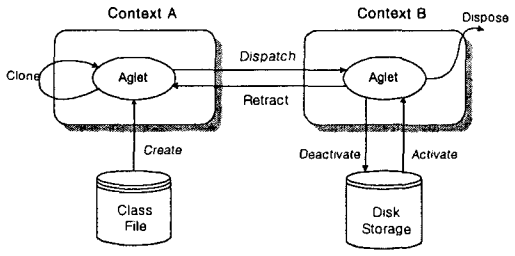

그림 3. aglet의 life-cycle 모델

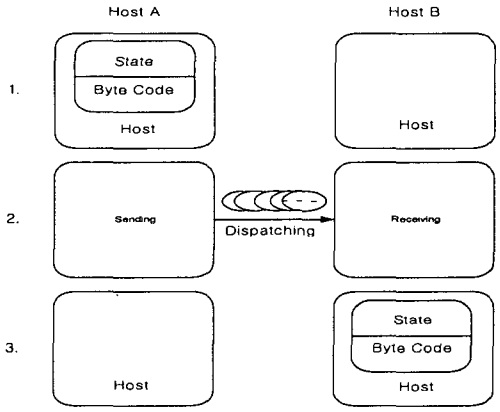

그림 4. Dispatching 되는 과정의 예

다음은 aglet의 이동에 핵심이 되는 dispatching과 retraction의 Java 코드에 대한 설명이다.

5.1 Dispatching

Aglets 프로그래밍 모델은 이벤트-기반으로 이루어진다. aglet의 dispatching이나 retraction 등의 이동 이벤트가 발생하였을 경우 MobilityListener라는 listener가 이를 감지하여 이에 대한 실행코드가 실행할 수 있도록 한다. Aglets의 이벤트 위임처리 모델은 aglet의 이동 이벤트가 발생했을 때 com.ibm.aglet.event 패키지 안의 클래스에서 이에 대한 처리를 하게 되며, 이 클래스 중에서 특히 추상화(abstract) adapter 클래스들은 이동

이벤트를 위한 listener 인터페이스를 구현한 것이다.

하나의 이벤트는 에이전트 시스템에서 aglet의 listener 패키지로 전달되어진다. 이들 listener 인터페이스는 그 인터페이스에 의해서 다루어지는 각각의 특별한 이벤트 형태에 대응하기 위해, 이벤트 소스에 의해서 호출되어지는 하나 혹은 몇 개의 메소드들로 정의되어져 있다. Dispatching에 필요한 메소드는 다음과 같다.

- URL에 의해서 정의되어지는 위치로 aglet을 dispatch한다. 그 위치의 context가 여러 개 있다면 특별히 "atp://www.host. com/public"과 같이 그 장소를 지정할 수도 있다.

public final void
Aglet.dispatch(URL destination)
예: dispatch(new URL("atp://www.host.com"));

- 이벤트 위임처리 모델은 dispatching을 효율적으로 제어할 수 있는 메소드를 제공한다. 위의 addMobilityListener는 이동 이벤트를 받을 수 있는 이동 listner를 추가한다. 이의 삭제를 위하여 removeMobility Listener를 사용한다.

public final void
Aglet.addMobilityListener
(MobilityListener listener)
public final void
Aglet.**removeMobilityListener**
(MobilityListener listener)

- aglet에 대해 dispatch()가 호출된 후 바로 onDispatching 메소드가 호출된다. 이 곳에 코드를 입력할 수 있는데, 예를 들면 현재 보낸 업무를 끝내고 다음에 보낼 준비를 할 수 있다. 이를 위한 인터페이스인 MobilityAdapter()를 통하여 이동 adapter의 인스턴스(instance)를 만들어야 한다.

public MobilityAdapter.**MobilityAdapter()**
public void MobilityAdapter.**onDispatching**
(MobilityEvent event)

- dispatch()의 실행이 종료되면 onArrival 메소드가 호출되며, 이어 run()이 실행된다. 이 onArrival 메소드는 목적지에 도착한 후 aglet을 초기화한다.

public void MobilityAdapter.**onArrival**
(MobilityEvent event)

그림 5는 앞에서 설명한 dispatching과 관련된 메소드들이 어떻게 동작하는가를 시간의 흐름(수평)에 따라 보여주고 있다. 그림의 Origin 부분은 dispatching 되기 전 aglet의 실행을 나타내며 Destination 부분은 목적지에서의 실행을 보여준다.

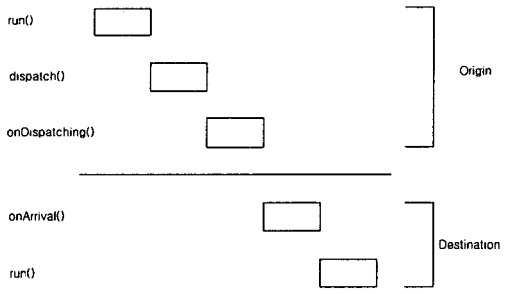

그림 5. aglet의 Dispatching 과정

다음 코드는 그림 5에 대한 한 예제 프로그램을 기술한 것이다.

```
public class DispatchingEx extends Aglet {
  boolean isRemote = false;
    //aglet이 dispatching되면 true
  public void onCreation(Object init) {
```

```
    addMobilityListener(
      new MobilityAdapter() {
        public void onDispatching
          (MobilityEvent e) {
          // 원하는 코드를 삽입
        }
        public void onArrival
          (MobilityEvent e) {
           isRemote = true; }
           // aglet이 목적지에 도착하면 true
           // 원하는 코드를 삽입
      }
    );
  }

  public void run() {
    //aglet 생성 후 실행 시작
    if(!isRemote) {
      try {
        URL destination =
          new URL("atp://www.dest.com");
          // Dispatch될 원격 호스트의 위치를 지정
        dispatch(destination);
          // 지정한 URL로 aglet을 dispatch
      } catch(Exception e) {
        System.out.println(e.getMessage());
      }
    } else { ... }
    }
  }
}
```

5.2 Retraction

Retraction은 dispatching과 반대되는 개념이다. 경우에 따라서는 aglet을 기다리기보다는 다른 호스트로부터 필요한 aglet을 요청할 필요가 있는데, 이 경우 retraction을 사용한다. 이 메소드는 다음과 같이 요청된 호스트의 위치와 retraction 되는 aglet을 지정하는 두 개의 매개변수를 가지고 있다.

Dispatching은 한 호스트로부터 다른 호스트로 byte code와 이와 관련된 state를 보낸다고 해서 능동적이라고 말하는 반면에 retraction은 다른 호스트로부터 필요한 byte code와 이와 관련된 state를 받는다고 해서 수동적이라고 말한다.

AgletContext.retractAglet
(URL contextAddress, AgletID aid)

이동 listener는 이 retraction을 위해 onReverting()과 dispatching에서 설명한 onArrival()이라는 두 개의 메소드를 지원하게 된다. 원격 aglet에 대해 retraction을 시도하게 되면 onReverting()이 호출되는데 이는 지정된 aglet이 retraction 될 것이라는 것을 경고하는데 사용될 수 있다. 또한 이 메소드는 retraction을 위하여 aglet을 준비하는 것을 허락하는데 override 된다. 그림 6은 aglet의 retraction되는 과정을 시간의 흐름(수평)에 따라 나타낸 것이다.

public void MobilityAdapter.**onReverting**
(MobilityEvent event)

그림 6의 retraction 과정을 보면, 우선 retraction을 요청한 Local 호스트에서 retractAglet() 메소드가 호출되면, Remote(원격) 호스트의 onReverting() 메소드를 원격 호출하게 된다. 이때 모든 원격 aglet들의 스레드는 모두 끝나게 된다. 이어 Local 호스트의 onArrival()과 run() 메소드가 실행된다. 이에 대한 Aglets 코드와 해설은 다음과 같다.

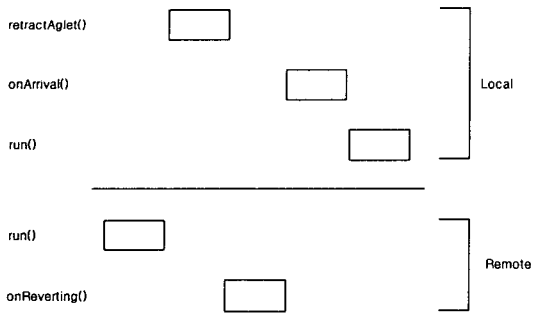

그림 6. aglet의 Retraction 과정

```
public class RetractingEx extends Aglet {
  public void run() {
    try {
      AgletProxy proxy =getAgletContext().
        createAglet(null,"RetractChild", null);
        // "RetractChild" aglet의 인스턴스 생성을
        // 위하여 aglet context를 사용
        // 생성된 aglet은 proxy에 그 참조를 반환
        // proxy는 차후 aglet이 dispatching될 때
        // 사용
      URL destination =
        new URL(atp://some.host.com);
        // 원격 호스트의 URL을 지정
      AgletID aid = proxy.getAgletID();
        // aglet의 식별자를 검색하고 aid에 저장
      proxy.dispatch(destination);
        //proxy는 aglet을 dispatching할 때 사용
getAgletContext().retractAglet
        (destination, aid);
// 원격 aglet 식별자로 원격 aglet을 retraction
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
public class RetractChild extends Aglet {
  // RetractChild는 Aglet 클래스로부터 상속받고
  // onCreation()과 run() 메소드로 override
  boolean _reverted = false;
    // aglet의 state 정보를 보유
    // onReverting()에서 원격 호스트에 aglet이
    // retraction되면 true
  boolean _dispatched = false;
    // aglet의 state 정보를 보유
    // onArrival()에서 원격 호스트에 aglet이
    // 도착하면 true
  public void onCreation(Object init) {
    // 이동 listener를 설치
    addMobilityListener(
      new MobilityAdapter() {
        public void
          onReverting(MobilityEvent e) {
          _reverted = true;
        }
        public void onArrival(MobilityEvent e) {
          _dispatched = true;
          if(_reverted)
            // 원하는 코드를 삽입
        }
      }
    );
  }
  public void run() {
    if(_dispatched)
      // 원하는 코드를 삽입
  }
}
```

6. 결 론

본고에서는 이동형 에이전트의 필요성, 실행 환경, 구현 언어 및 기존 시스템을 설명하였다. 또한 이동형 에이전트의 구현을 위하여 가장 널리 사용되는 언어인 Java를 이용한 대표적인 시스템인 Aglets을 기반으로, 이의 구현을 위한 기본 프로그램 예제를 설명하였다.

Java 언어는 네트웍 환경에서의 여러 장점으로 인하여, 이동형 에이전트 구현을 위한 최적의 언어로 평가받고 있다. 본고에서는 Java를 이용하여 이동형 에이전트에 의해 수행되는 제반 행위에 대한 코딩 예를 보임으로써, 이동형 에이전트의 구현을 위한 기본을 보였다. 따라서 이를 기본으로 보다 복합적이고 현실성을 고려한 이동형 에이전트 시스템의 구축이 가능할 것으로 판단된다.

앞에서 그 필요성에 대하여 충분히 언급하였지만, 인터넷의 사용이 보다 확산되면 될 수록 현재를 포함하여 앞으로도 에이전트의 필요성과 활용은 큰 비중을 차지할 것으로 예측된다. 특히 이동전화나 PDA와 같은 이동 통신 수단의 발달로 인하여 이동형 에이전트의 중요성은 더욱 커질 것으로 보인다.

그러나 이의 사용에는 선결 과제들이 있는바, 특히 향후 그 사용이 활성화될 것으로 기대되는 전자상거래 분야 등에서 활용되고자 하면 보안과 2.3절에서 설명한 주체에 대한 문제를 해결하여야 할 것이다.

참 고 문 헌

[1] Vu Anh Pham and Ahmed Karmouch, "Mobile Software Agents: An Overview," IEEE Comm. Magazine, pp.26-37, July 1998

[2] H. S. Nwana and M. Wooldridge, "Software agent technologies," BT Technol. J, pp.68-78, Oct. 1996

[3] D. Wong, N. Paciorek, and D. Moore, "Java-based Mobile Agents," CACM, pp.92-102, Mar. 1999

[4] S. Ichiro, "Survey on Mobile Agents," Japan Artificial Intelligence Society Review, pp.598-605, Jul.1999

[5] G. Cabri, L. Leonardi, and F. Zambonelli, "A Case Study in Mobile Agent Coordination," PDPTA '98 Int'l Conf., pp.810-817, 1998

[6] Robert L. Popp, Bohdan P. Maksymiuk and Michael R. Poreda, "Efficient Information Retrieval On the World Wide Web Using Adaptable And Mobile Java Agents," IEEE '98 Int'l Conf., pp.2219-2224, 1998

[7] Benzamin Falchuk and Ahmed Karmouch, "The Mobile Agent Paradigm Meets Digital Document Technology: Designing for Automous Media Collection," Multimedia Tools and Appl., pp.153-162, 1999

[8] Danny B. Lange and Mitsuru Oshima, *Programming and Deploing Java Mobile Agents with Aglets,* Addison-Wesley, 1998

[9] E. S. Lee, "A Construction and Application of Agent for Next Generation Computer Using," Int'l Biannual Conf. on Industrial Survival Strategy for Next Generation Software Technology, pp.115-140, 1996

[10] Peter Wayner, "Agents Away", BYTE, May 1994

[11] 김평중, 윤석환, "이동 컴퓨팅을 위한 이동 에이전트 시스템", 정보처리학회지, pp.67-75, Sep. 1997

[12] 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/ Architecture Synthesis(pAs97), p.17, Mar. 1997,

[13] D. B Lange and D. T. Chang, "IBM Aglets Workbench : Programming Mobile Agents in

Java," IBM White paper, http://www.trl.ibm.co.jp/aglets/whitepaper.html, Sep. 1996

[14] D. S Milojicic, M. Condict, F. Reynolds, D. Bolinger, and P. Dale, "Mobile Objects and Agents," 2nd USENIX Conference on Object Oriented Technologies and Systems : Distributed Object Computing on the Internet, Jun. 1996.

[15] H. Peine and T. Stolpman, "The Architecture of the Ara Platform for Mobile Agents," Proceedings of the First Int.l Workshop on Mobile Agents, p.12, Apr. 1997.

[16] M. Straber, J. Baumann and F. Hohl, "Mole: A Java Based Mobile Agent System," Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems, p.12, Jul. 1996.

[17] 최중민, "에이전트 개요와 연구 방향", 정보과학회지, pp.7-16, Mar. 1997

[18] Shoham Y., "Agent-oriented programming," Artificail Intelligence, Vol. 60, No.1, pp.51-92, 1993

임 준 식

- 1986년 인하대학교 전자계산학과 졸업 (학사)
- 1989년 University of Alabama at Birmingham (미) 전자계산학과 졸업 (석사)
- 1994년 Louisiana State University (미) 전자계산학과 졸업 (Ph.D.)
- 1995년~현재 경원대학교 전자계산학과 조교수
- 관심분야 : VLSI Design, Speech Recognition, Mobile Agent

황 대 훈

- 1977년 동국대학교 수학과 졸업(학사)
- 1983년 중앙대학교 대학원 전자계산학과(공학석사)
- 1991년 중앙대학교 대학원 전자계산학과(공학박사)
- 1983년~1985년 한국산업경제기술연구원(KIET) 연구원
- 1995년~1999년 경원대학교 전자계산소장
- 1987년~현재 경원대학교 교수
- 관심분야 : 멀티미디어 시스템, 인터넷 응용기술, 가상현실 등