정보
산업동향

# 모바일 에이전트 시스템 기술 동향

김 수 중*, 윤 용 익**

● 목 차 ●



## 1. 서 론

모바일 에이전트(Mobile Agent) 기술은 1994년 이후 클라이언트/서버 모델이 갖는 단점 즉, 지연 및 대역폭 문제와 네트워크 연결 해제의 취약점 등을 극복하기 위한 분산처리 시스템의 모델로서 급속히 발전을 시작하여 Agent Tcl, Aglets SDK, Ara, Concordia, Mole 등 많은 모바일 에이전트 시스템이 개발되었다.

모바일 에이전트 기술의 장점은 네트워크 부하와 트래픽, 그리고 대기시간의 감소이다. 수행할 실행코드와 데이터를 한 번에 이동하여 로컬 시스템에서 업무를 수행하므로 리얼타임 응답이 필요한 지역 분산시스템에 응용이 가능하므로 대기시간이 감소된다. 모바일 에이전트는 프로토콜을 캡슐화할 수 있기 때문에 네트워크 상에서 데이터가 움직일 때 프로토콜을 구현한 객체를 캡슐화하여 이동하는 것이 가능하다. 따라서 모바일 에이전트 사이의 독자적인 프로토콜 구현과 통신이 가능하다. 또한 자율적이고 비동기적으로 수행될 수 있으므로 접속 해제가 자주 발생하거나 회선상태가 좋지 않은 곳에서 효율성을 높일 수 있다. 이 외에도 동적인 적응성과 이기종 간의 수행 용이성 등의 장점이 있다.

모바일 에이전트 시스템이란 네트워크로 연결된 독립성이 높은 다수의 독립적인 소프트웨어 에이전트가 대등한 관계를 유지하며 하나의 시스템으로서의 기능을 수행하는 시스템으로 현재의 클라이언트 서버 시스템의 결점을 해결하는 중요한 분산처리 시스템의 대안으로서 인식되고 있다. 모바일 에이전트 시스템의 장점은 다양한 이기종 시스템을 쉽게 통합할 수 있다는 점과 상황 변화에 따라 프로그램이 동적으로 변화할 수 있기 때문에 유연성이 뛰어나다는 점이다

본 논문에서는 모바일 에이전트 시스템에 대한 표준화 동향을 살펴보고 공통적으로 요구되는 기본 요소들에 대하여 설명할 것이다. 그리고 몇 가지 모바일 에이전트 시스템들의 특성을 살펴보고 마지막으로 결론을 맺는다.

※ 본 논문은 2001년도 숙명여자대학교 교비연구비 지원으로 수행됨
* 숙명여자대학교 컴퓨터과학과 박사과정
** 숙명여자대학교 정보과학부 교수

## 2. 표준화

다양한 모바일 에이전트 시스템들이 등장하면서

이들간의 상호운용성(interoperability)을 지원하기 위한 미들웨어 연구와 표준화의 필요성이 대두되었다. 두 가지 주요 표준화는 Foundation for Intelligent Physical Agents(FIPA)[6]와 OMG Mobile Agent Facility(MAF)[7]이다.

FIPA는 1996년 형성되어 이질적인 에이전트들의 상호작용과 에이전트 기반 시스템에 대한 소프트웨어 표준안을 제시하였다. FIPA 표준안은 FIPA 97, 98을 거쳐 현재 FIPA 2000이 발표된 상태이다. FIPA 표준안의 핵심은 많은 벤더 플랫폼에 걸쳐 에이전트와 에이전트 시스템의 상호작용을 용이하게 하는 것이다. FIPA 표준안은 추상적 구조(Abstract Architecture), 에이전트 통신(Agent Communication), 에이전트 관리(Agent Management), 에이전트 메시지 전송(Agent Message Transport), 응용프로그램(Applications)의 분야로 세분화되어 있다.

MAF는 1998년 Object Management Group(OMG)에서 채택한 에이전트 시스템의 표준안이다. MAF의 중요한 목적은 서로 다른 에이전트 시스템 사이의 상호운용성이다. 상호운용성 증진을 위하여 MAF에서 표준화하고 있는 분야는 에이전트 관리(Agent Management), 에이전트 전송(Agent Transfer), 에이전트 및 에이전트 시스템 이름(Agent and Agent System Names), 에이전트 시스템 타입(Agent System Type), 로케이션 신택스(Location Syntax)이다. MAF 표준안에서 에이전트 통신은 CORBA의 객체 통신에 기반한다.

## 3. 모바일 에이전트 시스템의 기본 요구조건

### 3.1 에이전트(Agent)

에이전트는 개인이나 조직을 대신하여 자율적으로 행동하는 객체이다. 각 에이전트는 실행 스레드를 가지고 있어서 스스로의 의지에 따라 태스크를 수행할 수 있다. 모바일 에이전트는 실행을 시작한 시스템에 국한되지 않으며 컴퓨터 네트워크 상에서 에이전트를 수용할 수 있는 서버로 이동이 가능하다. 모바일 에이전트는 코드와 지속적인 상태 정보로 구성된다.

### 3.2 서버(Server)

에이전트 서버는 에이전트를 생성, 실행, 전송, 종료시킬 수 있다. 에이전트 서버는 이름과 주소로써 식별되며 하나 이상의 place를 포함한다. place란 에이전트 서버 내에 존재하고 하나 이상의 에이전트가 실행될 수 있는 컨텍스트이며 시스템 하부구조에 대한 접근을 제공한다.

서버는 모바일 에이전트 시스템의 견고성을 보장하고 신뢰성을 제공해야하며 시스템 고장에 따른 영향으로부터 에이전트를 보호해야 한다. 또한 서버는 모바일 에이전트 시스템 뿐만 아니라 에이전트와 관련된 상태 정보의 지속성을 보장해야 하고 트랜잭션 큐잉을 제공하여 에이전트가 목적지에 도착하는 것을 보장해야 한다.

### 3.3 모빌러티(Mobility)

모바일 코드 시스템에 기반한 모빌러티 메커니즘에는 두 가지 방법이 있다. 첫째, strong 모빌러티는 실행 유닛의 코드와 실행 상태 모두를 다른 연산환경으로 마이그레이션하는 것이다. 캡쳐된 실행 상태가 에이전트와 함께 전송되면, 목적지 서버는 마이그레이션이 시작된 시점부터 스레드를 재활성화할 수 있다. 목적지에서 상태가 복구되었을 때, 에이전트 코드는 제어 흐름을 적절히 감독할 수 있게 된다. 둘째, weak 모빌러티는 다른 연산 환경을 통해 코드를 전송하는 것이다. 이 경우 코드는 초기 데이터에 수반될 수 있으나 실행 상태 마이그레이션에는 영향을 미치지 않는다.

에이전트 모빌러티에 대한 다른 하나의 쟁점은 에이전트 코드 전송이다. 한 가지 방법은 에이전트가 이동할 때 모든 코드를 수반함으로써 이 코드를

실행시킬 수 있는 어떠한 서버 상에서도 에이전트가 작업을 수행하도록 하는 것이다. 일부 시스템은 코드를 전혀 전송하지 않고 목적지 서버 상에 에이전트 코드를 미리 설치하는 방법을 사용하기도 한다. 또 다른 방법은, 에이전트가 코드를 수반하지는 않으나 코드 베이스에 대한 참조를 포함하는 것이다.

### 3.4 통신(Communication)

에이전트는 작업을 수행하기 위해서 다른 에이전트, 서비스 제공자, 에이전트 소유자 등과 통신할 수 있어야 한다. 기본적인 통신 모드는 동기적 통신과 비동기적 통신이다. 동기적 통신에서는 결과가 돌아올 때까지 호출자가 블럭되고, 비동기적 통신에서는 호출자가 태스크를 계속 수행하면서 결과가 돌아왔는지에 대해 주기적으로 점검할 수 있다.

### 3.5 보안(Security)

모바일 에이전트 시스템에 관련된 보안 문제는 에이전트로부터의 호스트 보호, 다른 에이전트로부터의 보호, 호스트로부터의 에이전트 보호, 하부 네트워크 보호 등으로 나눠볼 수 있다.

- 에이전트로부터 호스트 보호: 호스트는 에이전트의 소유자를 인증하고, 이 인증에 따라 액세스와 자원 허용을 할당해야한다. 또한 모바일 에이전트를 전송하기 전과 영구적인 저장소에 저장하기 전에 에이전트를 암호화함으로써 보호해야한다.
- 다른 에이전트로부터의 보호: 에이전트는 다른 에이전트를 방해하거나 다른 에이전트의 자원을 사용해서는 안된다.
- 호스트로부터 에이전트 보호: 호스트는 에이전트를 간섭하거나 에이전트의 협조 없이 민감한 정보를 에이전트 외부로 유출해서는 안된다.
- 하부 네트워크 보호: 에이전트가 네트워크 자원을 과도하게 소비하지 않도록 해야한다. 예를 들어, 에이전트가 끊임없이 네트워크를 로밍하면서 계속 자신을 복제할 경우, 이러한 복제를 발견하여 종료시키는 방법이 필요하다.

## 4. 모바일 에이전트 시스템 동향

### 4.1 Agent Tcl

Agent Tcl[4]은 Dartmouth College에서 개발 중인 다중 언어 에이전트 시스템이다. D'Agent는 네 개의 레벨로 구성된다. 최저 레벨부터 열거하면 전송 메커니즘에 대한 인터페이스, 서버, 실행 환경, 에이전트이다.

Agent Tcl에서의 에이전트는 스크립트 언어로 작성된 프로그램으로서 수정된 인터프리터 상에서 수행된다. 서버는 수행되는 에이전트를 기억하여 그들의 상태에 대한 질의에 응답하며 에이전트 마이그레이션을 담당한다. 들어오는 에이전트를 받아 소유자의 신원을 인증하고, 인증된 에이전트를 적절한 인터프리터에게 전달한다. 외부로 나가는 에이전트에 대해서는 서버가 전송 메커니즘을 선택한다.

각 서버는 계층적인 이름공간(namespace)을 제공하고 에이전트들이 이 이름공간 안에서 서로에게 메시지를 보내도록 함으로써 통신을 가능하게 한다. 이름공간의 최상위 부분은 에이전트의 네트워크 위치에 대한 상징적인 이름이다.

에이전트는 내부 상태를 서버가 제공하는 비휘발성 저장소에 백업할 수 있다. 머쉰 고장 이벤트가 발생하면 서버는 비휘발성 저장소로부터 에이전트를 복구한다.

새로운 머쉰으로 이동하고자 하는 에이전트는 단일 함수를 호출한다. 이 함수는 에이전트의 완전한 상태를 자동적으로 캡쳐하고 목적지 머쉰의 서

버에게 이 상태 정보를 전송한다. 목적지 서버는 적당한 인터프리터를 구동하여 상태 정보를 실행 환경으로 로딩한 후, 에이전트가 중지된 정확한 지점에서 에이전트를 재시작한다. 네트워크를 통한 에이전트 전송은 일반적인 TCP/IP 소켓이나 전자메일을 사용하여 이루어진다.

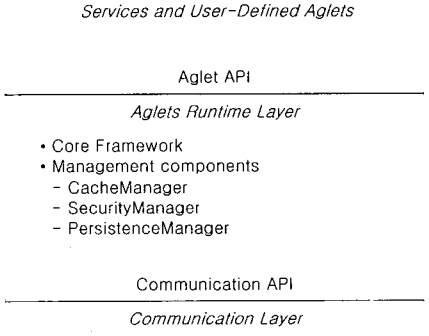
에이전트 통신을 위해서 에이전트들은 서버의 이름공간 안에서 서로에게 메시지를 전송한다. 메시지는 임의의 바이트 열로서 이벤트 메시지와 연결 메시지를 제외하고는 신택스나 시멘틱스가 미리 정의되어 있지 않다. 이벤트 메시지는 중요한 사건 발생에 대한 비동기적인 공지를 제공하고, 연결 메시지는 직접 연결 설정을 요청하거나 거부하는데 사용된다. 직접 연결은 에이전트 간의 명명된 메시지 스트림이며, 긴 상호작용에 대해서는 메시지 패싱보다 편리하고 효율적이다. 서버는 들어오는 메시지를 버퍼링하고, 나가는 메시지에 대해서 최상의 전송 메커니즘을 선택하며 연결 요청이 받아들여지면 명명된 메시지 스트림을 생성한다.

또한 에이전트는 직접 로컬 통신이나 AIDL (Agent Interface Definition Language)에 정의된 Agent RPC를 사용한 프로시저 호출을 통해 원격 통신을 할 수 있다.

## 4.2 Aglets SDK

Aglets Software Development Kit(ASDK)[1]은 일본의 IBM Tokyo Research Laboratory에서 개발되었다. ASDK를 구성하는 요소들을 살펴보면, 기본 기능을 제공하는 라이브러리인 Aglets Framework([그림 1])와 네트워크를 통한 에이전트 전송에 사용되는 Agent Transfer Protocol(ATP)가 있다. ATP는 분산 에이전트 기반 시스템을 위한 응용 계층 프로토콜이다. 또한 로컬 에이전트 서버인 Visual Agent Manager(Tahiti), Aglet 컨텍스트를 제공하는 Agent Web Launcher(Fiji)가 있다.

Aglet은 실행 중에 실행 컨텍스트로부터 다른 실행 컨텍스트로 자신을 디스패치할 수 있다. Aglet이 디스패치되었을 때, 해당 리스너(listener) 객체가 즉시 호출된다. 이것은 Aglet이 자신의 실행을 서스펜드하도록 하고, Aglet과 연관된 객체들을 정렬하여 목적지로 전송되도록 한다. 그런 다음 스레드는 종료된다. 수신 측에서는 자바 객체가 원점으로부터 수신된 데이터에 따라 재구축되어 새로운 스레드로 실행된다.




(그림 1) Aglets 프레임워크

Aglets는 ATP를 사용한다. ATP는 dispatch, retract, fetch, message 메소드를 정의한다. 이 프로토콜이 요청/응답의 형식을 수용하기는 하지만, 에이전트 간의 통신 스킴을 위한 어떤 규칙도 규정되어 있지 않다.

ASDK는 클래스캐쉬(classcache)와 함께 클래스로더(classloader)를 사용하여 에이전트의 코드를 전송한다. Aglet이 디스패치되면 Aglet을 수신한 ASDK 서버는 들어오는 클래스 정의들이 캐쉬에 있는 기존의 클래스들과 완전히 호환되는지를 검사하고, 충돌하는 클래스가 최소한 하나라도 있다면 새로운 클래스로더를 생성한다.

응용 프로그램이나 Aglet은 로컬 또는 원격 메시지 패싱에 의해 Aglet 객체들과 통신할 수 있다. ASDK는 now-type, future-type, oneway-type의 세 가지 메시지 패싱을 정의하고 있다. now-type은 동기

적인 메시징 방법으로 메시지 수신자가 메시지 처리를 완료하고 응답할 때까지 실행을 블록시킨다. future-type과 oneway-type은 모두 비동기적인 메시징 방법으로 현재 실행을 블록시키지 않는다. future-type 메시징에서는 송신자가 핸들을 계속 유지하는 반면, oneway-type 메시징에서는 송신자가 메시지 핸들을 유지하지 않고 수신자는 이 메시지에 대해 응답하지 않는다.

모든 Aglet 객체는 메시지 큐 객체를 갖는다. 들어오는 모든 메시지는 메시지 큐에 저장된 다음 하나씩 처리된다. ASDK에서 메시지는 보내어진 순서대로 도착하고 메시지의 종류에 따라 우선순위를 가질 수 있다.

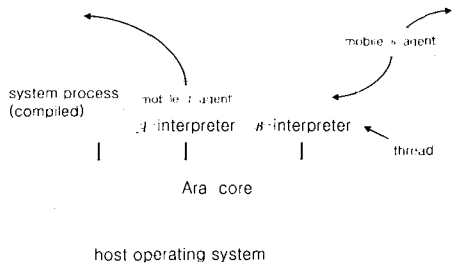
## 4.3 Ara

Ara[2]는 독일의 University of Kaiserslautern에서 개발한 모바일 에이전트를 위한 다중 언어 플랫폼이다. 에이전트는 각 프로그래밍 언어의 인터프리터에서 수행되고 공통 시스템 코아에 의해 제어되며 서비스를 제공받는다. Ara는 Tcl, C, C++를 지원하는데, 인터프리터가 해석할 수 있는 바이트 코드로 미리 컴파일된다.

Ara의 기본 컴포넌트는 코아라는 런타임 시스템으로 운영체제 상에서 단일 응용 프로세스처럼 수행된다. 코아 시스템은 시스템이 지원하는 프로그래밍 언어들을 위해서 다양한 인터프리터를 수행시킨다. 또한 코아 시스템은 에이전트 전송과 수신을 위한 통신 프로세스를 포함하고 있다. 시스템의 상주(stationary) 부분은 성능 측면에서 머쉰에 종속적인 코드로 컴파일된다.

Ara 서버는 많은 가상 place를 포함하는 코아 프로세스이다. 각 place는 현재 그 place에 있는 에이전트만 접근할 수 있는 서비스 포인트를 갖는다.

Ara agent는 시스템 호출을 사용하여 마이그레이션하며 네트워크를 통한 전송에는 일반적인 소켓, 전자메일, SSL을 사용한다. Ara의 시스템 구조는 다음의 [그림 2]와 같다.




(그림 2) Ara 시스템 구조

## 4.4 Concordia

Concordia[5]는 Mitsubishi Electric Information Technology Center America(MEITCA)에서 1997년에 개발을 시작하였고 자바 모바일 에이전트의 개발과 실행을 위한 프레임워크이다. Concordia를 구성하는 주요 요소는 에이전트 실행과 전송을 담당하는 서버와 원격 서버 관리를 위한 Administration 관리자이다.

Concordia 모바일 에이전트는 다른 노드들로 이동하면서 정보 에이전트(information agent)와 상호작용한다. 응용 프로그램 내의 에이전트들은 협동 유닛(collaboration unit)인 에이전트 그룹을 하나 이상 형성한다. 그룹에 속한 서로 다른 전문 에이전트들이 태스크를 동시에 수행한 후 협동을 통해 얻어진 산출물에 기반하여 결과를 서로 연관시키고 연산을 조정한다.

각 Concordia 서버는 모바일 에이전트에게 통합된 환경을 제공하는 다양한 모듈 컴포넌트로 구성된다([그림 3]).

- 에이전트 관리자(agent manager): 에이전트의 생성, 실행, 소멸을 관리한다.
- 큐 관리자(queue manager): 네트워크를 통한 에이전트 전송을 스케쥴링한다.
- 정보 에이전트(information agent): Concordia 서버는 하나 이상의 정보 에이전트를 통해 서비

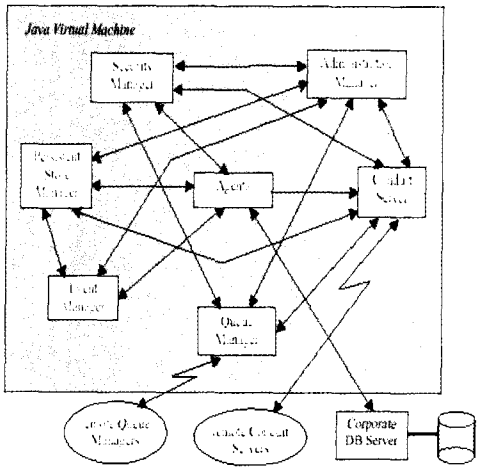
스를 제공한다.

- 영속적 저장 관리자(persistent store manager): 시스템 고장으로부터 에이전트 복구를 보장한다.
- 이벤트 관리자(event manager): 에이전트 그룹들의 협동을 지원한다.
- 보안 관리자(security manager): 로컬 보안 정책을 제어한다.
- 디렉토리 관리자(directory manager): 각 호스트 상에서 상호작용하고자 하는 응용 서버의 위치를 에이전트가 정할 수 있도록 한다.

각 서버는 큐 관리자를 통해 모빌러티를 제공한다. 큐 관리자는 store-and-forward 메커니즘을 사용하여 원격 호스트가 수신할 때까지 에이전트를 로컬 시스템 메시지 큐에 저장한다. 에이전트가 새로운 위치로 전송될 때 에이전트의 모든 내부 상태가 함께 전송된다. Concordia는 클래스로더를 지원하는데, 클래스로더는 바이트코드를 특별한 데이터 구조로 패키지화하고 이것은 에이전트와 함께 이동한다. 에이전트 전송은 자바 RMI와 SSL을 사용하여 이루어진다.

Concordia는 에이전트 간 통신을 위해 분산 이벤트(distributed events)와 에이전트 협동(agent collaboration)을 제공한다. 분산 이벤트는 이벤트 관리자에 의해 스케쥴링 및 관리된다. 에이전트가 선택된 이벤트를 수신하기 전에, 에이전트는 수신을 원하는 이벤트의 목록과 이벤트를 보낼 위치에 대한 참조를 이벤트 관리자에게 보내어 등록해야 한다. 에이전트 간 협동을 위해서는 각 에이전트가 그룹에 속해야 하며 특별한 협동점(collaboration point)에서 만나야 한다. 각 에이전트가 이 협동점에 도착하면, 에이전트는 연산 결과를 에이전트 그룹에 보내고 그룹의 모든 에이전트가 도착할 때까지 블럭된다. 에이전트 그룹은 각 에이전트의 연산 결과를 수집하고 그룹의 모든 에이전트들이 협동점에 도착하면 협동 메소드가 응용 프로그램의 분석 메소드를 호출하여 수집된 결과 집합을 전달한다.

Concordia는 strong 협동 모델과 weak 협동 모델을 지원한다. strong 협동에서는 각 에이전트가 그룹에 참여할 것을 요구하는 반면 weak 협동에서는 그룹의 일부 에이전트가 협동점에 도착하지 못하더라도 협동이 진행된다.




(그림 3) Concordia 시스템 데이터 흐름도

## 4.5 Mole

Mole은 1995년 독일의 University of Stuttgart에서 개발하였으며 자바로 구현된 최초의 모바일 에이전트 시스템이다. Mole 시스템은 자원 관리자(resource manager), 디렉토리 서비스(directory service), 에이전트와 place 및 메시지를 검사하는 그래픽 에이전트 모니터(graphical agent monitor), 자바 애플릿을 위해 WWW 서버로 사용되는 서버 엔진(server engine)으로 구성된다. Mole은 서버 엔진의 relay 컴포넌트를 사용하여, 로딩된 서버로만 연결되는 자바 애플릿의 제한성을 해결하였다.

Mole 시스템에서의 에이전트는 객체들의 클러스터로 설계되었고 에이전트 시스템 함수를 실행하는 데 사용되는 실행 환경 외에는 외부에 대해 참조하지 않는다. 에이전트는 정의된 통신 메커니즘

을 통해서만 다른 에이전트들과 통신할 수 있다.

에이전트 시스템은 place의 집합으로 구성되고 각 place는 에이전트 커뮤니티에 할당된다. place는 어떠한 서비스 집합에 대한 접근을 허용하거나 요금 정책을 구현한다. place는 실행 환경을 나타내는 서버 엔진으로서 에이전트들이 연산, 통신, 하부 시스템의 자원 및 서비스를 이용할 수 있는 장소이다. 이러한 자원과 서비스는 place 내부에서 시스템 에이전트로 나타내어지고 통신 메커니즘을 통해 사용자 에이전트에 의해 사용되어진다.

Mole은 자바 RMI를 사용하여 weak 마이그레이션을 제공한다. 에이전트가 DNS 형식의 대상 place 이름을 가지고 migrateTo 메소드를 호출하면 에이전트에 속한 모든 스레드는 서스펜드되고 새로운 메시지와 에이전트에 대한 호출은 수락되지 않는다.

에이전트에 대한 모든 미결정 메시지가 전달된 후에 그 에이전트는 활동 중인 에이전트 목록에서 삭제된다. 그런 후 에이전트는 객체 직렬화(serialization)을 사용하여 직렬화된다. 직렬화된 에이전트는 대상 로케이션으로 전송되어 다시 인스턴스화된다. 필요한 자바 클래스를 로컬에서 이용할 수 없을 경우, 대상 로케이션은 코드 서버나 소스 로케이션에게 이 클래스를 요청한다. 에이전트가 인스턴스화되면 새로운 에이전트 스레드가 재시작되며, 스레드가 에이전트의 제어를 담당하게 되면 곧이어 success 메시지가 소스 로케이션으로 전송된다. 이때 소스 로케이션은 에이전트에 속한 모든 스레드를 종료시키고 시스템으로부터 제거한다.

통신을 하고자 하는 에이전트는 실제 통신이 시작되기 전에 세션을 성립시켜야 한다. 세션 설정 후, 에이전트는 RMI나 메시지 패싱을 통해 상호작용할 수 있다.

지금까지 살펴본 모바일 에이전트 시스템들 몇 가지 측면에서 비교해 보면 [표 1]과 같다.

<표 1> 모바일 에이전트 시스템 비교

| | Agent Tcl | ASDK | Ara | Concordia | Mole |
|---|---|---|---|---|---|
| 플랫폼 | Unix | JDK1.1 | SunOS, Linux | JDK1.1 | JDK1.1 |
| 언어 | Tcl, Java, Scheme, Python | Java | Tcl, C, C++ | Java | Java |
| 자원 제어 | ○ | - | ○ | ○ | ○ |
| 마이그레이션 형식 | strong | weak | strong | weak | weak |
| 마이그레이션 메커니즘 | sockets, email | ATP, RMI | sockets, SSL | RMI, SSL | RMI |
| 코드 마이그레이션 | with data | code base | with data | with data | code server |
| 로컬 통신 | message, RPC | message | message | MI | MI, message |
| 전역 통신 | message, RPC | message | - | RMI | RMI, message |

## 5. 결 론

기존의 모바일 에이전트 시스템들은 구조와 구현 측면에서 다양한 형태를 갖추고 있기 때문에 그 표준화가 요구되었으며 FIPA와 MAF 등의 표준화 작업이 진행되고 있다. 특히 모바일 시스템 사이의 상호운용성을 지원하기 위한 MAF는 에이전트 관리와 전송에 관련된 메소드를 정의하고 에이전트 및 에이전트 시스템의 이름과 타입 등에 대한 신택스와 시멘틱스를 표준화하고 있다.

모바일 에이전트 활용의 확산을 위해 이러한 표준안의 채택은 반드시 필요한 것이다. 그러나 MAF는 CORBA나 DCOM 등의 기존 미들웨어와 마찬가지로 이기종 소프트웨어 컴포넌트간의 상호운용성을 주된 목적으로 하고 있기 때문에, 에이전트 간의 통신은 원격 프로시저 호출(Remote Procedure Call)과 같은 직접적인 통신을 기반으로 이루어진다. 직접적인 통신 방법은 네트워크 상의 지연과 신뢰성 문제를 야기할 수 있기 때문에 모바일 에이전트의 본질적인 장점인 효율적인 대역폭 사용과 비동기성 등을 충분히 활용할 수 없다.

모바일 에이전트는 코드와 데이터가 함께 이동하는 논리적 모빌러티의 특성을 가지고 있다. 많은 모바일 에이전트 시스템들이 자바 기반 기술을 사

용하여 모빌러티를 제공하고 있으나 일부 응용들은 그보다 더욱 유연적인 모빌러티를 요구한다. 예를 들어, 서버와 GSM 모바일 전화 사이의 대역폭이 9,600일 경우 서버로부터 많은 양의 자바 바이트코드를 다운로딩할 수 없다.

결론적으로, 앞서 언급한 통신 기법에 대한 발전적인 연구는 데이터 모빌러티를 증진시킬 수 있는 기반이 될 것이며, 자바 기술의 취약점을 해결할 수 있는 유연적인 모바일 코드 패러다임에 대한 연구가 진행되어야 할 것이다.

## 참고문헌


[1] D. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998

[2] H. Peine, T. Stolpman, The Architecture of the Ara Platform for Mobile Agents, First International Workshop on Mobile Agents, MA'97, April 1997

[3] J. Baumann, F. Hohl, K. Rothermel, M.Strasser, Mole - Concepts of a Mobile Agent System, Technical report 1997/15, Fakultaet Informatik, University of Stuttgart, August 1997

[4] R. S. Gray, Agent Tcl: A flexible and secure mobile-agent system, Proceedings of the Fourth Annual Tcl/Tk Workshop(TCL 96), 1996

[5] Concordia, Mobile Agent Computing --A White Paper, http://www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html, Mitsubishi Electric Information Technology Center of America, 1997

[6] Foundation for Intelligent Physical Agents(FIPA), http://www.fipa.org

[7] Mobile Agent Facility Specification(MAF), http://www.omg.org


## 저자약력


**김 수 중**

1995년 숙명여자대학교 전산학과 (이학사)
1998년 숙명여자대학교 전산학과 (이학석사)
1998년-현재 숙명여자대학교 컴퓨터과학과 박사과정
관심분야 : 분산 미들웨어, 모바일 에이전트 시스템, 액티브 네트워크



**윤 용 익**

1983년 동국대학교 통계학과 (이학사)
1985년 한국과학기술원 전산학과 (공학석사)
1994년 한국과학기술원 전산학과 (공학박사)
1998년-현재 숙명여자대학교 정보과학부 교수
관심분야 : 정보통신, 멀티미디어통신, 분산시스템, 실시간처리시스템, 분산미들웨어시스템, 분산 DBMS, 실시간 OS/DBMS