울산대학교 공학연구논문집 제31권 1호 pp. 381~395, 울산대학교.
Journal of Engineering Research Vol. 31, No. 1, pp. 381~395, 2000
University of Ulsan.

이기종간의 상호운용성을 지원하는 이동 에이전트 시스템

유양우* · 김진홍 · 문남두 · 안건태 · 박양수 · 이명준*
*컴퓨터 · 정보통신공학부

<요 약>

현재 대부분의 이동 에이전트 시스템들은 그들 자신의 구조를 가지며 서로 다른 방법으로 구현되어 있다. 이러한 에이전트 시스템들은 서로 다른 인터페이스를 제공하고 있어서 이기종의 에이전트 시스템에서 생성된 에이전트들은 상호 이동하여 작업을 수행할 수 없다. 이러한 문제를 해결하기 위하여 이동 에이전트 시스템간의 상호운용성 지원이 중요한 과제로 다루어지고 있다. *OMG(Object Management Group)*는 이질적인 에이전트 시스템간의 상호운용성을 증진시킬 목적으로 *MAF(Mobile Agent Facilities)* 명세를 제안하였다. MAF 명세는 MAFAgentSystem과 MAFFinder 두 개의 인터페이스로 구성되어 있으며 에이전트 관리, 코드의 이동성, 명명 규칙에 관한 기능을 제공한다.

본 논문에서는 OMG의 MAF 명세를 만족하는 SMART 이동 에이전트 시스템의 설계와 이의 구현에 대하여 설명한다. SMART 이동 에이전트 시스템의 구성은 에이전트를 실행시키는 환경을 제공하는 *플레이스(place)*, 에이전트의 라이프사이클과 플레이스를 감시하는 *모니터(monitor)* 그리고 에이전트에게 시스템의 자원을 할당하는 *자원관리자(resource manager)*로 이루어져 있다.

A Mobile Agent System Supporting Interoperability among Heterogeneous Mobile Agent Systems

Yu, Yang-Woo* · Kim, Jin-Hong · Moon, Nam-Doo
Ahn, Geon-Tae · Park, Yang-Su · Lee, Myung-Joon*
*School of computer Engineering Information Technology, University of ulsan

* 본 연구는 정보통신 진흥원의 '99년도 정보통신 우수시범학교 지원 사업의 지원으로 수행되었음

<Abstract>

Most current mobile agent systems adopted their own architectures, being implemented in various ways. Since those agent systems provide different interfaces, agents created in one mobile agent system could not move to other agent systems to perform their tasks. To solve this problem, *interoperability* becomes one of the important issues on mobile agent systems. The *OMG(Object Management Group)* proposed the *MAF(Mobile Agent Facilities)* specification for the interoperability among heterogeneous mobile agent systems. The MAF specification contains MAFAgentSystem and MAFFinder interfaces, and defines agent management, code mobility and naming rules.

In this paper, we describe the design and implementation of the SMART mobile agent system which satisfies the OMG MAF specification. The SMART system consists of Places that provide execution environments for agents, Monitor that monitors life-cycle of agents and places, and *Resource Manager* that allocates resources of system for agent.

1. 서론

네트웍 중심의 프로그래밍이나 어플리케이션들에 대한 관심은 인터넷 사용자 수의 증가와 WWW의 폭발적인 인기 때문에 최근 몇 년 사이 급진전하고 있다. 이에 부흥하여 이 같은 어플리케이션을 생성하기 위한 새로운 기술, 언어, 그리고 패러다임들이 쏟아져 나오고 있다[2].

*에이전트*라고 하는 것은 사용자를 대신해서 자발적으로 행동하는 컴퓨터 프로그램을 의미한다. 따라서 이동 에이전트라고 하면 컴퓨터 네트웍에서 어떤 계산을 수행하기 위하여 노드와 노드사이를 자율적으로 이동할 수 있는 사용자 대행 프로그램을 말한다. 그들의 일은 에이전트 프로그래밍에 의해서 결정되고, 그 응용은 인터넷 전자 상거래에서 실시간 디바이스 컨트롤, 과학적인 계산을 요하는 분산처리에 이르기까지 다양하다[1].

*이동 에이전트 시스템*의 기본적인 특성은 이동 에이전트를 인터넷상에 보내어 그들이 미리 설계된 경로를 따라 가거나 그들 자신이 모은 정보에 의해서 직접 동적으로 경로를 설정하여 돌아다닐 수 있도록 만든다. 이들 에이전트들은 그들의 임무를 달성했을 때 그 결과를 사용자에게 전달하기 위하여 그들의 홈 사이트로 돌아오게 된다[8].

이동 에이전트를 이용하는 기술은 기존의 통신 패러다임과는 달리 코드를 이동시켜 목적지 시스템에서 지역적으로 실행시키는 특성 때문에, 서버의 인터페이스를 바꾸지 않고 클라이언트의 다양한 요구를 융통성 있게 서비스 할 수 있으며, 또한 많은 양의 네트웍 트래픽을 줄일 수 있다. 이러한 장점을 제공하는 에이전트 기술은 현재 다양한 분야에서 널리 각광받고 있다.

현재, 다수의 이동 에이전트 시스템들은 그 구조와 구현이 매우 상이하다. 이러한 시스템들간의 차이는 에이전트 기술의 빠른 확산과 상호운용성(interoperability)[1]을 가로막는다. 하지만, 서로 다른 업체의 시스템들 사이의 상호운용성은 다가오는 서비스 시장의 요

구를 잘 이행하기 위하여 필수적인 사항이다. 상호운용성과 시스템의 상이성을 개선시키기 위하여 이동 에이전트 기술의 몇 가지 양상은 표준화되어야만 한다. 현재 이동 에이전트의 영역에서 가장 중추적인 표준화 단체는 MAF(Mobile Agent Facilities)를 제시한 OMG(Object Management Group, Inc)이다. OMG에서는 이동 에이전트 시스템의 상호운용성 기능에 대하여 표준 구조로 MAF 명세를 제안하였다[1]. MAF 표준은 1998년 새로운 OMG 기술로 채택하고 있다. 그 표준안은 에이전트 관리, 코드의 이동성, 명명 규칙과 같은 중요한 특성들로 이루어져 있다. 본 논문에서는 MAF 명세를 만족하는 이동 에이전트 시스템인 SMART 시스템을 Java 프로그래밍 언어를 이용하여 개발하였다. 이로써, 본 시스템은 상호운용성을 제공할 수 있으며, 시스템의 상이성을 해결할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 이동 에이전트 시스템간의 상호운용성을 제공하기 위하여 OMG에서 제안한 MAF 명세의 인터페이스에 대하여 소개하고, 각 메쏘드들이 동작하는 방법에 대해 살펴본다. 3장에서는 SMART 시스템의 전반적인 구조와 각 모듈의 역할에 대해 설명하고, 4장은 SMART 시스템을 이용한 에이전트 프로그래밍 예제를 보여준다. 5장에서는 최근 에이전트 시스템들의 경향과 그들의 시스템을 비교한다. 끝으로 6장에서는 결론 및 추후 연구 방향에 대하여 살펴본다.

2. OMG MAF 명세

2.1 상호운용성

이동 에이전트 기술에서 가장 중요한 목적은 다양한 에이전트 시스템 사이의 상호운용성이다. 에이전트 전송과 클래스 전송, 그리고 에이전트 관리에서 표준화가 된다면 상호운용성은 더욱더 쉽게 이루어질 수 있다 [1]. 이를 위하여 OMG에서는 이종의 이동 에이전트 시스템에 대하여 상호운용 가능한 인터페이스를 정의하여 MAF 명세를 작성하였다. MAF 명세의 구성은 에이전트를 생성하고, 목적지 에이전트 시스템으로 전송하는 기능을 제공하는 *MAFAgentSystem*과 명명(naming) 서비스를 지원하는 *MAFFinder* 두 개의 인터페이스로 이루어져 있다. SMART 시스템은 상호운용성을 지원하기 위하여 두 개의 인터페이스를 구현하였으며, 아래의 장에서 더 자세히 설명할 것이다.

2.2 MAFAgentSystem 인터페이스

MAFAgentSystem 인터페이스는 에이전트 관리 작업을 지원하는 메쏘드와 객체를 정의하고 있다 [1]. 주로 에이전트 시스템의 이름과 위치 정보를 검색하고, 에이전트를 받아들이는 작업을 한다. 이러한 메쏘드와 객체들은 에이전트 전송에 관한 기본적인 몇 가지 오퍼레이션을 제공한다.

- ***create_agent()*** : 에이전트 시스템은 원격 클라이언트의 요청에 의해 에이전트를 생성하기 위하여 이 메쏘드를 수행한다. 리턴 값은 생성된 에이전트의 실제 이름을 반환한다.
- ***fetch_class()*** : 이 메쏘드는 에이전트 시스템에서 에이전트가 작업을 수행할 때, 클

라이언트의 클래스 또는 코드를 불러올 수 있으며, 명시한 코드 베이스와 클라이언트로부터 클래스를 검색하는 데 주로 사용한다. 리턴 값은 하나 이상의 클래스 정의를 반환한다

- ***get_MAFFinder()*** : 에이전트와 플레이스 그리고 에이전트 시스템들을 검색하기 위하여 사용되며, MAFFinder에 대한 레퍼런스를 반환한다.
- ***receive_agent()*** : 에이전트 시스템은 에이전트를 받아들이고 인스턴스화 시키기 위하여 receive_agent()를 사용한다.
- ***get_MAFFinder()*** : 에이전트와 플레이스 그리고 에이전트 시스템들을 검색하기 위하여 MAFFinder의 레퍼런스를 반환한다.
- ***terminate_agent***() : 특정 에이전트의 수행을 중지시키는 메쏘드이다.

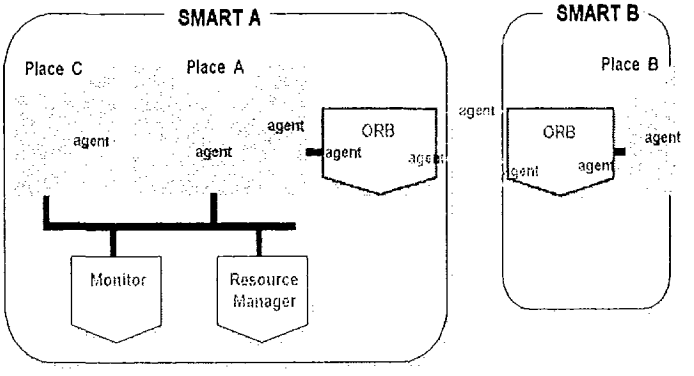
2.3 MAFFinder 인터페이스

에이전트와 플레이스(place), 그리고 에이전트 시스템의 동적인 이름과 위치(location)정보를 유지하기 위한 메쏘드를 MAFFinder 인터페이스에서 제공한다 [1].

- ***lookup_agent()*** : 파라미터에 명시한 에이전트의 위치를 반환한다. 이 메쏘드는 이름 또는 특정에이전트 프로화일로 에이전트를 검색할 수 있다.
- ***lookup_place()*** : MAFFinder에 등록된 플레이스의 위치를 구할 수 있다.
- ***lookup_agent_system()*** : MAFFinder에 등록된 에이전트 시스템의 위치를 찾을 수 있다.
- ***register_agent()*** : MAFFinder에 등록된 에이전트의 리스트에 에이전트를 추가시키는 메쏘드이다. MAFFinder 내의 이미 존재하는 에이전트 이름으로 호출된다면, 이 오퍼레이션은 관련 정보를 가장 최근에 호출된 정보와 대치시킨다.
- ***register_place()*** : 플레이스를 MAFFinder에 등록된 플레이스의 리스트에 추가시킨다.
- ***register_agent_system()*** : 에이전트 시스템을 MAFFinder에 등록된 에이전트 시스템의 리스트에 추가시킨다.
- ***unregister_agent()*** : MAFFinder에 등록된 에이전트의 리스트에 명시된 에이전트를 제거한다. 플레이스와 에이전트 시스템의 등록 해제는 ***unregister_place()***와 ***unregister_agent_system()***를 이용한다.

3. SMART 시스템의 구조

OMG MAF 명세를 만족하는 SMART 시스템은 에이전트를 실행하고 중지시키는 행동에 대하여 공통된 인터페이스를 정의함으로서 서로 다른 에이전트 시스템의 에이전트를 제어할 수 있는 상호운용성을 제공한다. SMART 시스템의 구성은 다섯 개의 모듈로 이루어져있다. 그 형태는 [그림 1]에서 자세히 설명하고 있다.



[그림 1] 전체적인 SMART 시스템 구조

에이전트를 실행시키고 종료시키는 환경을 제공하는 플레이스(place)와 에이전트의 라이프사이클과 플레이스의 실행을 감시하는 모니터(monitor), 그리고 에이전트에게 에이전트 시스템의 자원을 할당하고 회수하는 자원 관리자(resource manager)로 구성되어있다. 3장에서는 이들의 역할에 대하여 자세하게 살펴본다.

3.1 에이전트(SMART Agent)

이동 에이전트는 한 에이전트 시스템에서 다른 에이전트 시스템으로 이동하여 자발적으로 행동하는 컴퓨터 프로그램이다 [3]. SMART 시스템에서 에이전트는 자바 쓰레드로 동작하며, 에이전트가 자신의 작업을 모두 완료하면 미리 설계된 경로를 따라 이동하거나, 그들 자신이 모은 정보에 의해 직접 동적으로 경로를 설정하여 돌아다닌다.

3.1.1 에이전트 생성

클라이언트에서 에이전트를 생성할 때 *create_agent()* 메쏘드를 이용한다. 클라이언트는 자신이 에이전트를 생성하여 전송하는 일반적인 기능을 제공하지만, 목적지 에이전트 시스템에서 에이전트를 생성하여 에이전트를 실행시키는 기능 또한 제공한다. 이러한 기능을 원격 에이전트 생성(remote agent creation) 이라 부른다. 만약 에이전트가 목적지 에이전트 시스템이 아닌 다른 위치에 존재하면, code_base와 class_provider를 통해 그 위치를 얻을 수 있다.

```
public Name create_agent(
        Name            agent_name,     AgentProfile    agent_profile,
        byte[]          agent,          String          Place_name,
        Arguments       arguments,      ClassNameList   class_names,
        String          code_base,      MAFAgentSystem  class_provider)
```

[그림 2] 에이전트 생성 : create_agent()

3.1.2 에이전트 상태

이동 에이전트는 그들의 실행 환경인 플레이스에서 실행할 때, 다음과 같은 몇 가지 상태를 가질 수 있다. 에이전트의 상태는 "CfMAFRunning", "CfMAFSuspended", 또는 "CfMAFTerminated" 중에 하나의 상태를 갖고, get_agent_status()를 이용하여 에이전트의 상태를 구할 수 있다.

- **CfMAFRunning** : 에이전트가 자신의 작업을 현재 수행하고 있다면 그 에이전트는 "running" 상태를 가진다.
- **CfMAFSuspended** : 에이전트의 작업이 일시적으로 중단되었다면 그 에이전트의 상태는 "suspended"가 된다. 하지만 그 에이전트는 인스턴스를 가지고 있다, 그래서 다시 깨어났을 때, 자신의 작업을 계속할 수 있다.
- **CfMAFTerminated** : 에이전트가 자신의 실행을 완전히 마쳤다는 것을 의미한다.

3.2 플레이스(Place)

SMART 시스템에서 에이전트를 받아들이고 실행시키는 환경이 플레이스이다. 플레이스는 에이전트의 수행을 *suspend_agent(), resume_agent()* 그리고 *terminate_agent()*를 이용하여 제어할 수 있다. 플레이스는 하나의 에이전트 시스템에서 실행될 수 있으며, 또한 하나의 에이전트 시스템에 여러 개의 플레이스가 각각 독립된 실행환경으로 존재할 수 있다. 플레이스는 *createPlace()* 메쏘드를 이용하여 생성되고, 그 즉시 MAFFinder의 register_place()를 이용하여 플레이스를 등록시킨다. SMART 시스템에서는 클라이언트가 특정 목적지 에이전트 시스템 내에 임의의 플레이스를 동적으로 생성할 수 있는 기능을 제공한다.

3.2.1 플레이스 구조

플레이스는 에이전트를 실행시키는 환경으로서 데몬(daemon) 형태로 수행되고, 여러 개의 플레이스들은 독립적으로 실행되는 기능을 지원해야 한다. 이를 위해 SMART 시스템은 플레이스를 쓰레드로 구현하였다. 에이전트를 저장하기 위한 데이터 구조로는 ArrayList 타입으로 선언된 agentPool을 사용하고 있다. *putAgent()*와 *getAgent()* 두 개의 메쏘드를 이용하여 에이전트를 저장하고 저장된 에이전트를 하나씩 가져온다. 이 두 개의 메쏘드는 동기화를 위하여 자바의 동기화 메커니즘을 적용하였다.

3.2.2 에이전트 실행

에이전트는 자신이 수행해야할 코드를 가지고 여러 에이전트 시스템을 이동하며 그 코드를 실행시켜 자신의 작업을 수행한다. SMART 시스템에서 에이전트의 실행은 *쓰레드 그룹(thread group)*을 이용하여 에이전트가 생성한 다수의 쓰레드를 효율적으로 관리할 수 있도록 하였다 [4]. 쓰레드 그룹은 에이전트 시스템에서 유일한 식별자를 가지며, 이동 에이전트는 쓰레드 그룹으로부터 하나의 쓰레드를 할당받아 자신의 작업을 실행한다. 에이전트마다 하나의 쓰레드 그룹이 생성되어 에이전트가 실행 중에 발생하는 모든 쓰레드는 같은 그룹 내에서 관리된다.

3.2.3 에이전트 전송

SMART 시스템의 이동 에이전트는 미리 설계된 경로를 따라 이동하거나, 그들 자신이 모은 정보에 의해 직접 동적으로 경로를 설정하여 돌아다닌다 [3]. 이동은 목적지 에이전트 시스템이 결정되면, 목적지 에이전트 시스템으로 이동하기 위하여 현재의 에이전트 시스템에게 *leaveSmart()* 메쏘드를 이용하여 전송 준비를 요청한다. 그리고 에이전트 클래스와 상태는 네트웍을 통해 전송하기 위하여 객체를 바이트 스트림으로 변환시켜 주는 *objectToByte()*를 호출한다. 목적지 에이전트 시스템의 레퍼런스는 MAFFinder의 *lookup_agent_system()*을 이용하여 Dest_Smart의 위치 정보를 얻어내고, 이 정보를 이용하여 CORBA 명명(naming) 서버와 접속하여 객체의 레퍼런스를 구할 수 있다. 마지막으로 에이전트 전송은 *Dest_Smart.receive_agent()* 메쏘드를 호출함으로서 완료된다.

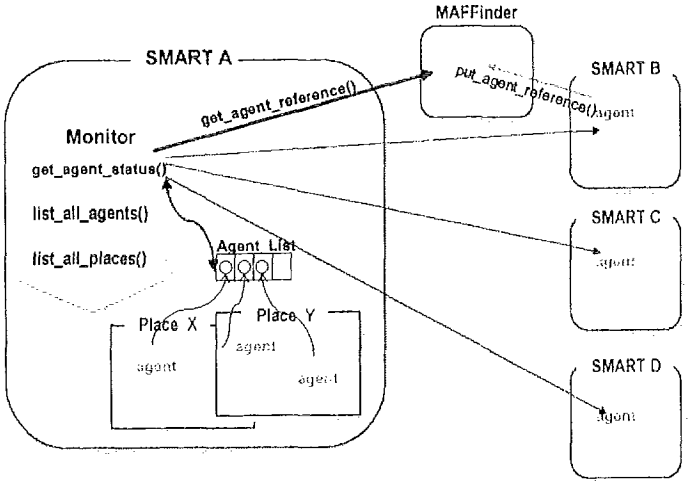
```
public void receive_agent(
        Name            agent_name,     AgentProfile    agent_profile,
        byte[]          AgentMessage,   String          Place_name,
        ClassNameList   class_names,    String          code_base,
        MAFAgentSystem  agent_sender)
```

[그림 3] 에이전트 전송 : receive_agent()

목적지 에이전트 시스템에 도착한 에이전트는 인증 과정을 거친 후 에이전트 클래스와 상태를 역 직렬화(deserialize) 시키고, 에이전트는 해당 플레이스의 agentPool에 저장되어 자신의 계정(identity)에 따른 자원을 할당받아 실행한다.

3.3 모니터(Monitor)

SMART 시스템이 기동될 때, 모니터는 하나의 데몬 형태로 자신의 수행을 시작한다. 모니터의 주된 임무는 크게 이동 에이전트의 감시와 플레이스를 감시하는 기능을 제공한다. 또 다른 기능으로, 원거리에 있는 에이전트 시스템에게 특정 플레이스에 주어진 이름으로 에이전트를 만들 수 있도록 요청할 수 있다. 자세한 내용은 다음과 같다.



[그림 4] SMART 시스템의 모니터의 동작

3.3.1 에이전트 관리

이동 에이전트들은 주어진 연산을 수행하기 위하여 한 노드에서 노드로 광범위한 영역을 돌아다닌다 [2]. 이러한 에이전트들은 특정 노드에서 수행 도중 시스템 또는 다른 요인으로 인해 그들의 수행이 중단될 수 있다. 에이전트 시스템은 기본적으로 에이전트의 라이프사이클 정보를 유지해야 한다. SMART 시스템에서는 *get_agent_status()*를 이용하여 각 목적지 에이전트 시스템에서 수행중인 에이전트의 상태를 파악할 수 있는 기능을 제공한다.

에이전트 시스템은 현재 시스템 내의 에이전트들의 이름과 수를 알고 있어야 한다. 이를 위하여 현재 SMART 시스템에서 수행중인 에이전트들은 Agent_List에 그 이름과 수를 기록하고, *list_all_agent()*를 이용하여 그 값을 구할 수 있다. 모니터는 MAFFinder를 이용한 위치 정보와 Agent_List 값을 이용하여 에이전트를 효율적으로 관리할 수 있다.

3.3.2 플레이스 관리

SMART시스템에서 플레이스는 하나의 시스템 내에 여러 개의 플레이스가 독립적으로 수행될 수 있도록 구현하였다. 모니터는 현재 에이전트 시스템 내에 정상적으로 동작하는 플레이스의 이름과 수를 저장하고 있는 count_place 속성을 제공한다. 그리고 *list_all_places()* 메쏘드를 이용하여 SMART 시스템 내의 모든 플레이스를 검색할 수 있다.

3.4 자원 관리자(Resource Manager)

SMART 시스템에서 자원 관리자는 이동 에이전트가 에이전트 시스템의 자원을 사용하려고 할 때, 요청한 자원에 대하여 접근 권한에 따라 자원 접근을 결정한다. 자원 관리자는 이동 에이전트나 다른 에이전트 시스템이 해당 에이전트 시스템의 특정 자원에 대한 접근 권한을 가진 경우, 요청한 자원을 사용할 수 있는 자원의 레퍼런스를 넘겨주는 반면, 인증 되지 않은 자원 접근을 요청을 할 경우, 내부 보안 예외 상황을 발생 시켜 자원에 대한 접근을 막는다 [6].

SMART 시스템의 자원에 대한 접근은 JDK1.2의 보안 관리자(SecurityManager)를 이용

하여 관리한다 [16]. 또한, 요청한 에이전트 또는 에이전트 시스템이 특정 자원에 대한 접근 권한을 가지고 있는지에 따라 자원 관리자는 자원을 관리한다. 자원 관리는 자바의 Policy 추상클래스를 상속받아 구현한 SmartPolicy 객체를 이용한다. SmartPolicy 내의 PermissionCollection에는 에이전트가 수행하는데 필요한 자원을 명시한다. 에이전트는 이러한 정보를 갖고 목적지 에이전트 시스템으로 이동하여 자원에 대한 사용 승인을 얻을 수 있다. 사용 승인의 종류와 그에 관련된 행위는 다음과 같다.

```
java.io.FilePermission              : File read/write/delete/execurt
java.net.SocketPermission           : Socket resolve/connect/listen/accept
java.lang.RuntimePermission         : getClassLoader/setContextClassLoader/
                                      setSecurityManager/setIO/stopThread/modifyThread/
                                      loadLibrary/queuePrintJob/accessDeclaredMembers
java.util.PropertyPermission
```

[그림 5] SMART 자원 허가(Permissions)

SMART 시스템의 자원 접근 정책은 계정을 기초로하는 액세스 제어 정책(identity-based access control policy)을 적용하였다 [11]. 이는 외부의 인증된 에이전트가 에이전트 시스템으로 들어오면 에이전트의 계정과 권한에 따라 다른 수준의 자원을 할당시켜 주는 정책이다. 본 시스템에서 에이전트는 다른 에이전트 시스템에서 자신의 작업을 수행하기 위하여 [그림 5]에서 정의한 자원 허가를 이용하여 Policy 객체를 생성한다. 목적지 에이전트 시스템에서 자원 관리자는 이러한 정보를 참조하여 에이전트가 수행하는데 필요한 자원을 계정 등급에 따라 할당하여 준다.

에이전트의 계정이 목적지 에이전트 시스템에 등록되어 있지 않다면, 이 에이전트의 수행은 JDK1.2의 모래상자(Sand Box) 메커니즘이 적용되어 극히 제한된 자원만을 할당받아 자신의 작업을 실행하게 된다 [16].

4. SMART: 에이전트 프로그래밍

4.1 클라이언트 프로그램

SMART 시스템의 에이전트 생성과 전송에 관련된 클라이언트 측면의 코드를 보여주고 있다.

```
public class SmartClient {
  static org.omg.CORBA.ORB              orb;
  static MAFFinder                      mafFinder;
  static MAFAgentSystem_Impl            mafAgentSystem;
  static MAFAgentSystem                 dest_mafAgentSystem, agentSystem;

  public static void main(String[] args) {
  //      ORB      Initialization      ----------
```

```
        orb      = org.omg.CORBA.ORB.init(args, props);
        client   = new SmartClient();
    // Create MAFAgentSystem_Impl Object ------------------------------
        mafAgentSystem = new MAFAgentSystem_Impl(orb);

    // Get MAFFinder Reference ----------------------------------------
        mafFinder = mafAgentSystem.getMAFFinder();
    // Get location of SMART_alpha agent system -----------------------
        locations = client.lookUp_agentSystem("SMART_alpha");
        dest_mafAgentSystem = mafAgentSystem.getObejctReference(locations[0]);
    // Create Agent!! -------------------------------------------------
        Name tempn = client.createAgent();
    }
}
```

[그림 7] SMART: Client 프로그램

4.2 에이전트 프로그램

다음의 코드는 SMART 시스템에서 에이전트에 해당하는 코드를 보여주고 있다.

```
public class Agent implements Runnable, Serializable {
    // Constructor ----------------------------------------------------
    public Agent(String name, int age, LinkedList path) {
        agentName       = name;
        agentAge        = age;
        agentPath       = path;
    }
    // Agent Behavior -------------------------------------------------
    public void run() {
          for (int i = 0; i < 10 ; i++) {
              this.agentAge++;
          }
          if(agentPath.size() != 0) {
              next = (String) agentPath.get(host);
              agentPath.remove(host);
              gotoNextHost(next);    // Go to next destination -----
          }else {
                :
    }
    public void gotoNextHost(String nextHost) {
    // Get location of nextHost ---------------------------------------
        dest_mafAgentSystem       =       mafAgentSystem.getObejctReference(nextHost);


    // Serialize Agent Object -----------------------------------------
        agent = mafAgentSystem.objectToByte(this);
```

```
  // Agent Moving ------------------------------------------
      dest_mafAgentSystem.receive_agent( AgentName, simpleAgentProfile, agent,
    place_name,
                                  class_names, code_base, current_mafAgentSystem);
  }
}
```

[그림 8] SMART: Agent 프로그램

4.3 서버 프로그램

SMART 시스템의 서버측 코드는 다음과 같다.

```
public class Smart {
    public static void main(String[] args) {
            :

// ORB and BOA Initialization ----------------------------------
      orb     = org.omg.CORBA.ORB.init(args, props);
              boa     = ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);

// Create MAFAgentSystem_Impl Object ---------------------------
      mafAgentSystem      =      new      MAFAgentSystem_Impl("Smart_alpha");

      boa.impl_is_ready(null);
    }
}
```

[그림 9] SMART: Server 프로그램

4.4 플레이스 프로그램

SMART 시스템에서 플레이스는 에이전트를 받아들이고 실행시키는 환경을 제공한다. 다음은 플레이스의 코드를 설명하고 있다.

```
public class Place implements Runnable {
      :
    public void run() {          // Place Behavior ----------------------
      while(true) {
        Agent agent = getAgent();
        Thread agentThread = new Thread(agent);
        agentThread.start();    // Execute Agent ------------------------
      }
    }
// Put a Agent in the agentPool ------------------------------
    public synchronized void putAgent(Agent anAgent) {
```

```
        agentPool.add(anAgent);
        notify();
    }
// Get a Agent ------------------------------
    public synchronized Agent getAgent() {
        if(agentPool.size()>0) {
                return get();
        }
        else {
                wait();
                return get();
        }
    }
    public synchronized Agent get() {
        Agent agent = (Agent) agentPool.get(0);
                agentPool.remove(0);
                return agent;
    }
}
```

[그림 10] SMART: Place 프로그램

5. 관련 연구

현재 이동 에이전트에 관한 연구는 활발히 진행되고 있다. 다양한 에이전트 시스템들은 에이전트의 이동성을 지원하기 위하여 제공하는 기법 또한 서로 다르다. 이 장에서는 지금까지 개발된 시스템들의 특징을 설명하고, 각 시스템에서 사용한 이동 에이전트의 이동성 기법에 대하여 비교한다.

IBM에서 개발한 에이전트 시스템으로는 *Aglets*이 있다 [8]. 프로그래밍 언어는 Java를 이용하여 개발하였으며, 에이전트 이동성은 "weak" 속성을 가진다. Aglets API는 통신환경으로서 "context"의 개념을 제공한다. aglet의 context는 몇 가지 기본적인 서비스를 제공한다. Aglets은 두 개의 이주(migration) 오퍼레이션을 제공한다. 하나는 "dispatch"로서 독립적인 코드를 파라미터에 명시한 context로 이동시킨다. 이 기법은 비 동기적인 속성을 가진다. 이와 대칭적인 오퍼레이션은 "retract"로서 코드를 가져오고, retract가 실행되었던 context로 돌아오도록 요구하는데 사용된다. 이는 동기적인 속성을 가진다.

미 캘리포니아 대학에서 개발된 에이전트 시스템인 Java-To-Go이다 [4]. 시스템 환경은 Java 프로그래밍 언어를 사용하여 개발하였으며, 에이전트 시스템의 기본적인 특성을 지원하는 비교적 간단한 시스템이다. 에이전트 이동성은 "weak" 속성을 가진다. 상호운용성은 지원하지 않는다. 특징은 에이전트 실행 환경인 "Hall"을 여러 개 수행할 수 있으며, 이를 관리하는 하나의 "Community" 서버에는 여러 개의 "Hall"이 수행된다. Community 서버는 또 다른 기능으로 에이전트 실행 시 필요한 클래스를 원격 에이전트 시스템으로부터 동적으로 가져올 수 있다.

*Ajanta*는 인터넷 상에서 이동 에이전트를 이용하여 애플리케이션을 프로그래밍 하기 위

한 Java 기반의 시스템이다 [5]. 미네소타 대학에서 개발하였으며, 다양한 기술을 적용한 시스템이다. 먼저 에이전트가 서버의 자원을 액세스할 때, 프록시(proxy) 기반 제어 기법이 사용된다. 또한 에이전트의 순회 패스를 프로그래밍하기 위하여 이주 패턴(migration pattern)의 개념을 사용하였다. 에이전트 이동성은 "weak" 속성을 지원하고 상호운용성은 제공하지 않는다.

*Grasshopper*는 독일의 GMD Focus 연구소에서 개발한 에이전트 시스템으로써 시스템 환경은 Java 프로그래밍 언어를 사용하여 개발하였다 [12]. 대표적인 특징은 OMG의 이동 에이전트 표준을 적용하여 구현한 첫 번째 에이전트 시스템이다. 에이전트 이동성은 "weak"속성을 가진다, 그리고 OMG MAF 명세를 따라 구현한 시스템이므로 이기종간의 상호운용성을 지원한다. Grasshopper 시스템은 RMI, Socket, IIOP 등 다양한 통신 채널을 제공한다.

<표 1> 에이전트 시스템 비교

에이전트 시스템	상호운용성	코드 이동성	보안 정책
Aglets	○	weak	Yes(MAC/MIC)
Java-To-Go		weak	No
Ajanta		weak	Yes (Proxy Object, RSA)
Grasshopper	○	weak	Yes (SSL/X.509 certificate)
SMART	○	weak	Yes (SSL, JDK SecurityManager)

6. 결론 및 추후 연구 과제

에이전트 패러다임은 자신의 코드를 이동시켜 목적지 시스템에서 실행시키는 특성을 제공하는 관계로, 인터넷 환경에서 분산 애플리케이션을 개발하는데 현재 널리 이용되고 있다. 하지만 대부분의 이동 에이전트 시스템들은 매우 다른 구조로 구현되어 있어 이종의 시스템에서 생성된 에이전트의 실행이 지원되지 않고 있다. 또한 이러한 시스템들간의 차이는 에이전트 기술의 빠른 확산과 상호운용성을 가로막고, 에이전트 시스템의 적용을 어렵게 하고 있다.

본 논문에서는 서로 다른 에이전트 시스템간의 상호운용성을 위하여 OMG의 MAF 명세를 만족하는 SMART 시스템의 설계와 구현에 대하여 설명하였다. MAF 명세는 MAFAgentSystem과 MAFFinder 두 개의 인터페이스로 이루어져 있다. MAFAgentSystem 인터페이스는 에이전트를 생성하고, 전송하고, 실행시키는 오퍼레이션을 정의한다. MAFFinder 인터페이스는 에이전트와 플레이스, 그리고 에이전트 시스템을 등록시키고, 등록된 것을 해제시키는 오퍼레이션 그리고 이들을 검색하는 오퍼레이션을 정의하였다. 이 두 개의 공통된 인터페이스를 이용하여 이기종간의 상호운용성을 지원할 수 있다. 그 결과 MAF 명세를 만족하는 이질적인 에이전트 시스템간의 통신이 가능하며, 다른 에이전트 시스템에서 생성된 에이전트도 수행할 수 있게 되었다.

SMART 시스템은 기능 별로 플레이스, 자원 관리자, 보안 관리자 그리고, 모니터의 네 개의 모듈로 이루어져 있다. 플레이스는 쓰레드 그룹을 이용하여 에이전트의 수행을 효율

적으로 관리할 수 있으며, 에이전트 전송 규칙은 에이전트 생성 시 필요한 클래스의 이름 목록을 전송하고 에이전트 실행 도중에 필요한 클래스들은 "On-demand" 방법으로 수행하는 기법을 사용하여 더 적은 네트웍 트래픽을 제공한다. 보안 관리자는 OMG MAF에서 요구하는 4가지 정책을 SSL을 기반으로 하여 간결하게 구현하였다. 그리고, 에이전트가 에이전트 시스템에게 자원을 요청 시 자원 관리자는 에이전트의 권한을 검사하여 자원을 할당하거나 거부한다. 모니터는 이동 에이전트와 에이전트를 실행시키는 플레이스가 정상적인 수행을 하고 있는지를 감시한다.

추후 연구 과제로서, SMART 시스템은 현재 에이전트 또는 에이전트 시스템들 간의 일대일 통신만을 지원하고 있다. 이러한 제한된 통신 수단을 극복하기 위하여 객체의 공유 개념을 제공하는 ObjectSpace를 사용할 것이다 [20]. ObjectSpace는 중앙 집중적인 공유공간 구조를 가지는 튜플 스페이스(Tuple space)를 분산된 공유공간 구조로 개선시킨 시스템으로서 튜플 스페이스가 가지는 단점을 해결하였다. ObjeceSpace는 서버의 실패나 네트웍의 분할(partition)이 발생하여도 신뢰성 있는 서비스를 제공하는 시스템이다. SMART 시스템에서 통신 수단으로 ObjectSpace를 사용하면 통신의 효율성을 향상시킬 수 있는 장점을 가진다.

참고문헌

[1] Mobile Agent System Interoperability Facilities Specification, OMG Inc, 1998. 3.
[2] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, "Understanding Code Mobility", IEEE Transaction On S/W Engineering, Vol. 24, NO. 5, May 1998.
[3] Neeran M. Karnik, Anand R. Tripathi, "Design Issues in Mobile Agent Programming Systems", University of Minnesota Minneapolis, June 1998.
[4] William Li, David G. Messerschmitt, Java-To-Go Mobile Agent System, University of California at Berkeley, 1998
[5] Anand R. Tripathi, Neeran M. Karnik, Manish K. Vora, Tanvir Ahmed, and Ram D. Singh, "Ajanta - A Mobile Agent Programming System", 1998.
[6] Neeran M. Karnik, "Security in Mobile Agent Systems", PhD thesis, University of Minnesota, October 1998.
[7] Gunter Karjoth, Danny B. Lange, Mitsuru Oshima, "A Security Model For Aglets", IEEE Internet Computing, 1997.7.
[8] B.Venners, "The Architecture of Aglets", JavaWorld http://www.javaworld.com/javaworld/ jw-04-1997/jw-04-hood.html, April 1997.
[9] Rocco De Nicola, Gian Luigi Ferrari, Rosario Pugliese, "KLAIM: A Kernel Language for Agents Interaction and Mobility", IEEE Transactions On SE, VOL. 24, NO. 5, May 1998.
[10] IBM Aglets Workbench-Home Page, 1999. URL address: http://www.trl.ibm.co.jp/aglets/
[11] C. Baumer, M. Breugst, S. Choy, T.Magedanz, "Release 1.2 Basics and Concepts", Grasshopper, February 1999.
[12] S. Choy, T.Magedanz, "Grasshopper Technical Overview", IKV++ GmbH, February 1999.
[13] ObjectSpace Inc. "VOYAGER Security 3.2 Developer Guide", http://www.objectspace.com/products /voyager, 1999.
[14] Tom Walsh, Noemi Paciorek, David Wong, "Security and Reliability in Concordia", IEEE 1060-345/98, 1998.
[15] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, "Concordia: An Infrastructure for Collaborating Mobile Agents", In Mobile Agents: First International Workshop, Vol. 1219, 1997.
[16] Krishna Sankar, "Java 1.2 Class Libraries Unleashed Vol I, II", Sams, 1999.
[17] Antonella Di Stefano, Lucia Lo Bello, Corrado Santoro, "Naming and Locating Mobile Agents in an Internet Environment", IEEE 0-7803-5784-1/99, 1999.
[18] T. Taka, T. Mizuno, T. Watanabe, "A Model of Mobile Agent Services Enhanced for Resource Restrictions and Security", IEEE 0-8186-8603-0/98, 1998.
[19] T. Finin, Y. Labrou, Y. Peng, "Mobile Agents Can Benefit from Standards Efforts on Interagent Communication", IEEE Communications Magazine, July 1998.
[20] 안건태, 문남두, 정현락, 유양우, 이명준, "JACE 그룹통신시스템을 이용한 신뢰성 있는 공유객체공간의 개발", 한국정보과학회 99가을 학술발표논문집(III) p218~220, 1999.