SMART : OMG의 MAF 명세를 지원하는 CORBA 기반의 이동 에이전트 시스템

유 양 우[†] · 김 진 홍[††] · 구 형 서[†††] · 박 양 수[††††] · 이 명 재[†††††] · 이 명 준[††††]

요 약

오늘날 이동 에이전트 기술은 네트워크 트래픽을 효과적으로 줄이고, 서버의 인터페이스를 바꾸지 않고 클라이언트의 다양한 요청을 융통성 있게 서비스할 수 있는 기술로 여겨지고 있다. 그러나 대부분의 이동 에이전트 시스템들은 그들 자신의 구조를 적용하고 다른 방법으로 구현되어, 특정 *이동 에이전트 시스템*에서 생성된 이동 에이전트는 다른 에이전트 시스템에서는 사용될 수 없다. 이러한 문제를 해결하기 위하여 OMG(Object Management Group)는 이질적인 에이전트 시스템간의 *상호운용성(interoperability)*을 증진시킬 목적으로 이동 에이전트와 에이전트 시스템간의 공통된 개념적인 모델에 대하여 기술한 *MAF(Mobile Agent Facilities)* 명세를 제안하였다.

본 논문에서는 OMG MAF 명세를 만족하는 자바 기반의 SMART 시스템에 관하여 기술한다. *SMART* 시스템은 MAF 명세에 따른 이동 에이전트간의 상호운용성을 지원하고, MAF 명세에서 권장하는 보안 모델을 기반으로 한 독자적인 보안 기능을 제공한다. 또한 보다 안정된 에이전트 시스템의 운영을 위하여 자체적인 에이전트 및 에이전트 시스템의 *영속성(persistence)*과 *예외처리*를 지원한다.

SMART : A CORBA-based Mobile Agent System Supporting the OMG MAF Specification

Yang-Woo Yu[†] · Jin-Hong Kim[††] · Hyeong-Seo Koo[†††] · Yang-Su Park[††††] · Myeong-Jae Yi[†††††] · Myung-Joon Lee[††††]

ABSTRACT

As of now, the *mobile agent* technology is considered to effectively reduce the network traffic and flexibly process various requests of clients without modifying the interface of servers. Since most mobile agent systems have adopted their own architecture, implemented in different ways, mobile agents created in a *mobile agent system* are not useful in other mobile agent systems. To solve the problem, the OMG (Object Management Group, Inc) proposed the *MAF (Mobile Agent Facilities)* specification suggesting a common conceptual model of mobile agents and mobile agent systems for the interoperability between heterogeneous mobile agent systems.

In this paper, we describe a mobile agent system named *SMART*, which satisfies the OMG MAF specification, written in Java. The SMART system supports interoperability with other mobile agent systems implemented according to the MAF specification, and also provides a security mechanism based on the security model recommended by the MAF specification. In addition, to support the robustness of the agent system, the facilities of exception handling and agent persistence are provided.

키워드 : 이동 에이전트(Mobile Agent), 이동 에이전트 시스템(Mobile Agent System), 상호운용성(Interoperability), 영속성(Persistence)

1. 서 론

네트워 중심의 프로그래밍이나 어플리케이션들에 대한 관심은 인터넷 사용자수의 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[1].

*에이전트*라고 하는 것은 사용자를 대신해서 자발적으로 행동하는 컴퓨터 프로그램을 의미한다. 따라서 *이동 에이전트*라고 하면 컴퓨터 네트워에서 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[2].

*이동 에이전트 시스템*의 기본적인 특성은 이동 에이전트를 인터넷상에 보내어 그들이 미리 설계된 경로를 따라 가

※ 본 연구는 정보통신진흥원의 2000년도 정보통신 우수시범학교 지원사업의 지원으로 수행되었음.

†준 회 원 : 울산과학대학 컴퓨터정보학부
††정 회 원 : 울산대학교 대학원 컴퓨터정보통신공학부
†††준 회 원 : 울산대학교 대학원 컴퓨터정보통신공학부
††††정 회 원 : 울산대학교 컴퓨터정보통신공학부 교수
†††††준 회 원 : 울산대학교 컴퓨터정보통신공학부 교수

논문접수 : 2000년 2월 15일, 심사완료 : 2001년 3월 7일

거나 그들 자신이 모은 정보에 의해서 직접 동적으로 경로를 설정하여 돌아다닐 수 있도록 만든다[3]. 이들 에이전트들은 그들의 임무를 달성했을 때 그 결과를 사용자에게 전달하기 위하여 그들의 홈 사이트로 돌아오게 된다[4].

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[5].

현재, 다수의 이동 에이전트 시스템들은 그 구조와 구현이 매우 상이하다. 이러한 시스템들간의 차이는 에이전트 기술의 빠른 확산과 상호운용성(interoperability)[2]을 가로막는다. 하지만, 서로 다른 업체의 시스템들 사이의 상호운용성은 다가오는 서비스 시장의 요구를 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[2]. MAF 표준은 1998년 새로운 OMG 기술로 채택하고 있으며, 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. 본 시스템은 MAF 명세의 표준 인터페이스(에이전트 시스템의 기능, 에이전트 위치 추적과 에이전트 전송 기능 등)구현과 이를 효과적으로 지원하기 위한 이동에이전트 시스템 모델을 제시하고 구현하였다. 또한 보다 안정된 시스템의 운영을 위하여 SMART 시스템은 *에이전트 예외 처리*와 *에이전트 영속성 (persistence)* 기능을 제공한다.

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 그들 시스템을 비교한다. 6장에서는 SMART 시스템에서 에이전트 생성, 이동, 그리고 추적 기능 등 시스템 구현 모습을 보여 준다. 끝으로 7장에서는 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[2]. 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[2]. 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, 클라이언트의 클래스 또는 코드를 불러올 수 있으며, 명시한 코드 베이스와 클라이언트로부터 클래스를 검색하는데 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 아들이고 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.

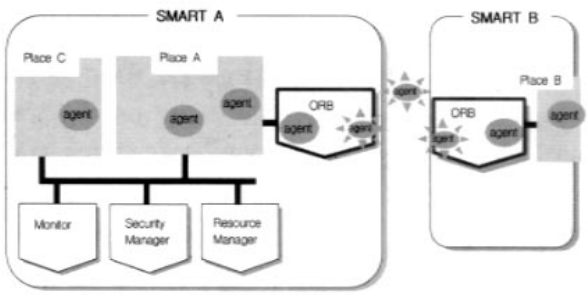
2.3 MAFFinder 인터페이스

에이전트와 플레이스(place), 그리고 에이전트 시스템의 동적인 이름과 위치(location)정보를 유지하기 위한 메쏘드를 MAFFinder 인터페이스에서 제공한다[2].

- ***lookup_agent()*** : 파라미터에 명시한 에이전트의 위치를 반환한다. 이 메쏘드는 이름 또는 특정 에이전트 프로파일로 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.
- 시스템을 MAFFinder에 등록된 에이전트 시스템의 리스트에 추가시킨다.
- 등록된 에이전트의 리스트에 명시된 에이전트를 제거한다. 플레이스와 에이전트 시스템의 등록 해제는 ***unregister_place()***와 ***unregister_agent_system()***를 이용한다.

3. SMART 시스템의 구조

MAF 명세를 분석하면 크게 에이전트를 생성, 중지, 종료시키는 에이전트 관리 기능과 MAFFinder를 이용한 에이전트 추적(tracking) 기능 그리고 에이전트 전송에 관한 기능을 정의하고 있다. 그리고 이동 에이전트 시스템이 갖추어야 할 요건[6]으로 에이전트의 실행환경과 보안사항을 제시하고 있다. 따라서 MAF 명세를 지원하는 효율적인 이동 에이전트 시스템은 앞에서 설명한 요건들을 효과적으로 지원하기 위한 구조를 가지고 있는 것이 바람직하다. SMART 시스템은 효율적인 이동 에이전트 시스템을 구축하기 위하여 에이전트를 실행시키고, 중지, 종료시키는 *플레이스(place)*, 에이전트를 추적하고 감시하는 *모니터(monitor)*, 에이전트의 안전한 전송을 위한 *보안관리자(security manager)*, 그리고 MAF 명세에서 제시하지 않았지만 이동 에이전트로부터 에이전트 시스템을 보호하기 위한 *자원관리자(resource manager)*로 구성된 네 개의 모듈로 전체시스템을 구성하였다.



(그림 1) 전체적인 SMART 시스템 구조

또한 MAF 명세에서 구체적으로 정의하지 않은 여러 가지 세부적인 기능을 각 모듈에 추가시켜 이동 에이전트 시스템의 기능을 향상시켰다. 네 개의 모듈로 구성된 SMART 시스템의 구조는 (그림 1)에서 설명하고 있다.

*플레이스(place)*는 이동 에이전트의 실행 환경을 제공하는 모듈이다. 이러한 기능을 효과적으로 지원하기 위하여 본 시스템은 각각의 플레이스를 병행성을 가진 쓰레드로 동작하게 하였다. 그리고 각 플레이스 내에서는 에이전트의 저장과 실행의 동기화를 효율적으로 지원하게 하였다.

*모니터(monitor)*는 에이전트의 생명주기(lifecycle)와 플레이스의 실행을 감시하는 기능을 담당하고 있다. 현재 시스템 내에 있는 이동 에이전트의 감시와 자신이 생성하여 여행중인 에이전트를 감시하는 기능 모두를 지원하고 있다. 시스템 내의 이동 에이전트 감시는 에이전트의 로그 기능을 지원하고 있으며, 다른 시스템 내에 있는 이동 에이전트의 감시는 다른 모니터와의 상호작용을 통한 통신채널을 이용하여 효과적으로 지원한다.

*자원 관리자(resource manager)*는 이동 에이전트 시스템이 갖추어야할 가장 중요한 기능 중에 하나로서 이동 에이전트에게 에이전트 시스템의 자원을 할당하고 회수하는 기능을 제공하고 있다. MAF 명세에서는 정의되어 있지 않고, 각 이동 에이전트 시스템 구현 시 그 개발자에게 맡기고 있다. SMART 시스템에서는 일반적인 보안정책을 지원하는 Java 자체의 보안정책을 지양하고, 독자적인 자원접근정책을 제시하고 있다. 본 시스템의 자원접근정책은 관리자입장의 자원접근 서비스를 정의한 클래스(Admin_Service class), 일반적인 사용자를 위한 자원접근 서비스(User_Service class), 가장 낮은 자원 접근 서비스(Gypsy_Service clsass)를 제공하여, 이동 에이전트의 자원접근등급에 따라 각각의 서비스를 이용하는 효율적인 자원접근정책을 제공하고 있다.

마지막으로 *보안 관리자(security manager)*는 이동 에이전트와 이동 에이전트 시스템의 보안기능을 제공한다. MAF 명세에서는 이동 에이전트 시스템이 지켜야할 보안사항으로 기밀성, 무결성, 인증, 재실행 탐지 네 가지 사항을 요구하고 있다. SMART 시스템은 안전한 소켓 계층(Secure Sockets Layer : SSL)[7]이 이동 에이전트 시스템을 구현시 표준화된 프로토콜로서 널리 사용되고 있으므로 이를 이용하여 기밀성, 무결성, 인증의 세 가지 사항을 만족시켰으며, 네 번째 사항인 재실행탐지는 SMART 시스템에서 제공하는 시스템 로그기능을 이용하여 지원하도록 하였다.

그리고, SMART 시스템은 이동에이전트 시스템이 갖추어야 할 추가적인 기능에 대한 해결 방안을 제시하여 효율적이고 안정된 기능을 제공하고 있다.

3장에서는 이들의 역할에 대하여 자세하게 살펴본다.

3.1 에이전트(SMART Agent)

이동 에이전트는 한 에이전트 시스템에서 다른 에이전트 시스템으로 이동하여 자발적으로 행동하는 컴퓨터 프로그램이다 [8]. 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. 에이전트에 대한 예제 프로그램은 4.3장에서 설명하고 있다.

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()

agent_name은 전역적으로 유일한 이름을 할당해야 한다. 에이전트가 정상적으로 생성되면 에이전트의 이름을 반환하고, 즉시 MAFFinder 클래스의 *register_agent()* 메쏘드를 이용하여 에이전트를 등록시킨다.

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[9]. 자세한 프로그램 코드 내용은 4.5장의 (그림 11)에서 설명할 것이다.

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[10]. 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[8]. 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.

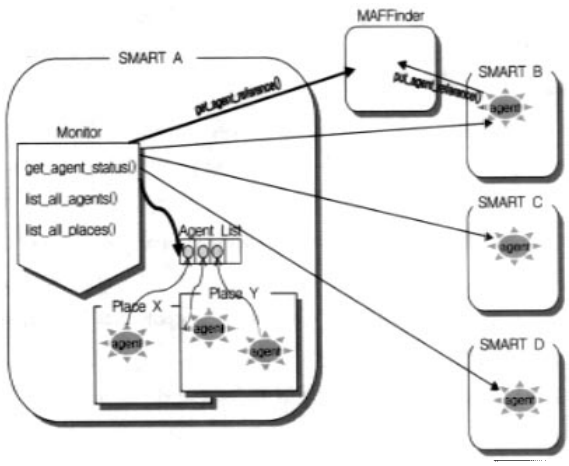
```
public void receive_agent(
  Name              agent_name,    AgentProfile  agent_profile,
  byte[]            AgentMessage,  String        Place_name,
  ClassNameList     class_names,   String        code_base
  MAFAgentSystem    agent_sender)
```

(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 [1]. 이러한 에이전트들은 특정 노드에서 수행 도중 시스템 또는 다른 요인으로 인해 그들의 수행이 중단될 수 있다. SMART 시스템은 *get_agent_status()*를 이용하여 각 목적지 에이전트 시스템에서 수행중인 에이전트의 상태(Execute, Suspended, Terminated)를 파악할 수 있는 기능을 제공한다. 그리고 자신이 생성한 에이전트에 대하여 원격으로 실행 상태를 변경할 수 있다.

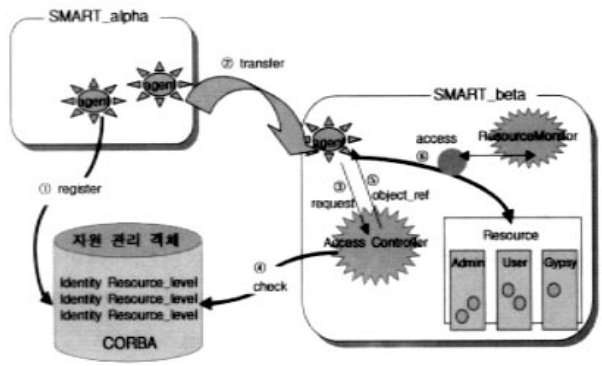
에이전트 시스템은 현재 시스템 내의 에이전트들의 이름과 수를 알고 있어야 한다. 이를 위하여 현재 SMART 시스템에서 수행중인 에이전트들은 Agent_List에 그 이름과 수를 기록하고, *list_all_agent()*를 이용하여 그 값을 구할 수 있다. 모니터는 MAFFinder를 이용한 위치 정보와 Agent_List 값을 이용하여 에이전트를 효율적으로 관리할 수 있다.

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 시스템의 자원에 대한 접근은 독자적으로 자원 관리자(ResourceManager) 클래스를 구현하여 자원 접근 정책을 적용하고 있다. 자원관리자 추상클래스를 상속받아 서비스에 따른 세 가지 클래스 Admin_Service, User_Service, Gypsy_Service를 구현한다. 각 클래스는 자원등급에 맞추어 에이전트가 수행할 수 있는 모든 저 수준의 API를 정의하고 있다.



(그림 5) SMART 시스템의 자원 접근 정책

SMART 시스템에서 이동 에이전트가 가질 수 있는 자원 등급은 Admin, User, Gypsy 세 가지로 분류된다. 에이

전트는 계정에 따른 자원 등급과 자신이 사용할 수 있는 서비스 객체의 메쏘드 정보를 가지고 에이전트 시스템에 도착한다. 에이전트 시스템에서는 이러한 정보를 원격 호스트에서 유지하고 있는 자원 관리 객체를 통해 자원 등급을 비교하여 등급이 같거나 작으면 그에 따른 자원을 할당하여 준다. 이 자원 관리 객체는 CORBA를 이용하여 구현하였으며, 저장하고 있는 정보는 계정과 그와 관련된 자원등급 그리고 사용하고자 하는 객체와 메쏘드 정보를 유지하고 있다. 이를 관리하는 관리자(Manager)는 계정에 따른 자원 등급을 상향시키거나 조절할 수 있는 권한을 가진다. SMART 시스템은 이동 에이전트가 사용할 수 있는 객체만을 사용하고 있는가를 감시하는 자원모니터(ResourceMonitor)가 있다. 이러한 기능의 구현은 에이전트가 동적 클래스 로더를 이용할 때 그 정보를 알아낼 수 있다.

3.5 보안 관리자(Security Manager)

OMG MAF 명세에서 에이전트가 오퍼레이션을 호출하거나 다른 에이전트 시스템으로 이동하고자 할 때, 네트웍 통신 보안의 품질에 대한 요구 사항을 명시한다. 요구 사항은 네 가지로 이루어져 있다.

- *기밀성(Confidentiality)* : 에이전트는 데이터를 안전하게 보낼 수 있는 통신 채널을 요구해야 하며, 또한 암호화의 정도를 명시할 수 있어야한다.
- *무결성(Integrity)* : 에이전트는 네트웍 통신 중에 데이터의 변조 또는 침입에 의한 수정을 탐지할 수 있는 무결성 검사를 요구할 수 있다.
- *인증(Authentication)* : 에이전트는 전송이 시작되기 전에, 목적지 에이전트 시스템의 인증을 요구 해야 한다.
- *재실행 탐지(Replay detection)* : 에이전트는 통신 세션 중에 에이전트의 중복을 예방하는 재실행 탐지 알고리즘을 사용할 수 있도록 에이전트 시스템에게 요구할 수 있다.

SMART 시스템의 보안 관리자는 *안전한 소켓 계층(Secure Sockets Layer : SSL)*을 기반으로 위의 네 가지 요구 사항을 구현하였다. SSL은 대칭 또는 비대칭적인 암호화 방법을 실제 사용할 수 있도록 작성한 산업 표준 프로토콜로서 기밀성과 데이터 무결성 그리고 클라이언트/서버의 상호 인증을 제공한다[11]. SMART 시스템에서는 *Protekt 3.0 SSL*[12] 패키지에서 제공하는 API를 이용하여 OMG MAF 명세에서 요구하는 세 가지 사항을 만족시켰다. 네 번째 요구 사항인 재실행 탐지는 에이전트의 이름 속성에 계정과 권한 정보를 에이전트 시스템의 로그(log) 파일과 비교하여 구현하였다. 다음은 SSL을 이용한 SMART 시스템의 Communicator와 Proxy 객체에 대한 예를 보여 주고 있다.

```
// Communicator.java
    SSLServerSocket ss = new SSLServerSocket(port);
    while(true) {
        SSLSocket s = (SSLSocket)ss.accept();
        ProxyObject po = new ProxyObject(s);
        po.start();
    }
// ProxyObject.java
    public void run() {
        try {
                        s.startHandshake();
                        s.waitForHandshake();
                        InputStream is = s.getInputStream();
                        OutputStream os = s.getOutputStream();
        } catch(Excetpion e) {
                        e.printStackTrace();
        }
        Agent_Message agent_Message = (Agent_Message)is.read();
    }

// ProxyClient.java
    public void run() {
        SSLSocket s = new SSLSocket(host, port);
        s.startHandshake();
        s.waitForHandshake();
        OutputStream os = s.getOutputStream();
        InputStream is = s.getInputStream();
        os.write(agent_Message);
        os.flush();
    }
```

(그림 6) SSL API를 이용한 프로그래밍

3.6 시스템의 견고성을 지원하기 위한 기능

MAF 명세는 에이전트 생성, 이동 그리고 에이전트 시스템과 이동에이전트의 위치 추적 기능 등 상호운용성을 지원하기 위한 기본적인 기능에 대하여 기술하고 있다. 그러나 이동에이전트 시스템이 상호운용성을 지원하고 보다 안정적인 시스템으로 구축하기 위해서 추가적인 기능이 요구된다. 추가적인 기능으로는 에이전트와 에이전트 시스템의 영속성, 에이전트 예외 처리 기능이 있다. 에이전트 시스템은 자신의 정보(실행 중인 에이전트, 플레이스)를 저장하고 종료하여, 종료 이전의 정보를 복구할 필요가 있다. SMART 시스템은 에이전트 시스템의 영속성과 예외를 발생한 에이전트를 저장하는 저장소로 JavaSpace[13]를 이용한다.

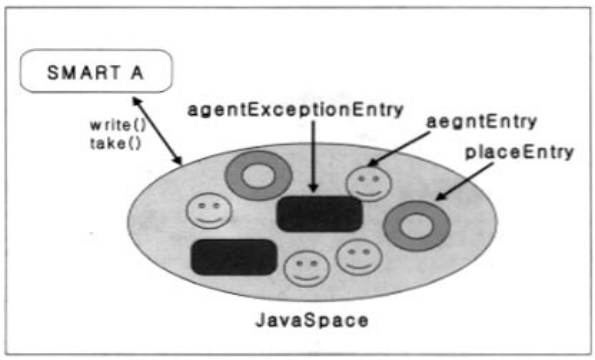
JavaSpace는 분산 객체의 영속성과 데이터 교환을 지원하는 Jini[14]서비스로써, 표준 인터페이스를 통하여 net.jini.core.entry.Entry 형태의 객체를 저장(write)하고 구할(take) 수 있는 기능과 저장된 객체는 필드들이 탐색 패턴으로 설정된 템플릿을 사용하여 탐색 기능 등을 제공한다. SMART 시스템의 영속성은 분산 객체의 영속성을 지원하는 JavaSpace의 표준 인터페이스와 JavaSpace에서 객체의 존재 시간을 지정하

는 Lease 개념을 이용하여 보다 쉽게 지원된다.

그리고, JavaSpace의 하부 기능을 제공하는 Jini는 네트워크상의 지능형 기기들이나 소프트웨어들이 동적인 상호작용을 지원하는 기술이다. Jini는 접속과 동시에 각 기기들의 중재자 역할을 수행하는 연합체(Loopup Service)를 형성하고 이 연합체를 통하여 서비스를 등록하고 클라이언트로부터 요구된 서비스를 지원한다. SMART 시스템은 Jini 시스템상의 에이전트 시스템의 영속성을 위한 서비스로써 JavaSpace를 이용하고 있다.

3.6.1 에이전트 시스템의 영속성(persistence)

SMART 시스템은 자신의 환경에서 현재 실행중인 에이전트와 플레이스에 대한 정보를 영속적으로 저장하는 기능을 JavaSpace를 이용하여 제공한다. 에이전트 시스템의 영속성은 에이전트 예외 처리와 에이전트 시스템의 정상 및 비정상 종료를 위하여 필요하다. SMART 시스템은 Jini 서비스인 JavaSpace에 에이전트 시스템의 정보를 위한 agentEntry, placeEntry와 에이전트 예외처리를 위한 agentExceptionEntry, 그리고 시스템 상태를 저장을 위한 systemStatusEntry형태로 저장한다. JavaSpace의 write()를 이용하여 저장된 agentEntry와 placeEntry는 이동에이전트와 플레이스 객체를 가지고 있으며, 각각의 이름을 인자로 JavaSpace의 take()를 이용하여 복원된다.



(그림 7) SMART 시스템 Persistence 기능

```
import net.jini.core.entry.*;
import net.jini.entry.AbstractEntry;

public class agentEntry extends AbstractEntry {
    public String agentName;
    public Agent agent;
    public agentEntry( String inAgentName,
                       Agent inAgent)
    {
        agentName = inAgentName;
        agent = inAgent;
    }
}
```

(그림 8) JavaSpace에 저장되는 Entry 예

이동 에이전트와 이동 에이전트 시스템을 저장하기 위한 저장소인 JavaSpace는 SMART 시스템마다 존재한다. SMART 시스템이 에이전트를 JavaSpace에 저장하는 시기는 에이전트가 생성될 때와 다른 시스템에서 에이전트가 이동되었을 때이며, 삭제하는 시기는 에이전트의 실행이 완료된 후와 에이전트 예외 처리 결과에 따른다. 그리고 플레이스를 JavaSpace에 저장하는 시기는 플레이스의 상태가 변경될 때(플레이스 생성, 새로운 에이전트 실행, 에이전트 삭제)이며, 삭제되는 시기는 시스템이 플레이스를 종료할 때이다.

SMART 시스템이 재 시작할 때, 시스템의 종료 상태를 고려하여 다음과 같은 과정을 거친다.

① SMART 시스템을 초기화한다.(ORB, 명명(Naming)서비스, MAFFinder, 모니터 등)
② JavaSpace에서 플레이스를 가져와서 실행시킨다.
③ 정상 종료 여부를 파악한다. (systemStatusEntry 참조)

- 정상 종료인 경우, 모든 에이전트를 실행시킨다.
- 비정상 종료인 경우, 에이전트의 자원접근 레벨에 따라 처리한다.
  - 시스템 자원을 접근하지 않는 에이전트는 바로 실행시킨다.
  - 시스템 자원을 접근하는 에이전트는 예외를 발생시킨다.

3.6.2 이동 에이전트의 예외처리 방법

이동 에이전트는 에이전트 시스템의 자원에 대한 접근이 거부되는 경우, 또는 이동하려는 목적지 에이전트 시스템을 찾지 못하는 경우, 그리고 에이전트 시스템이 비정상 종료하는 경우 예외 상황을 발생시킬 수 있다. 이때, SMART 시스템은 발생된 예외 상황을 에이전트 자원 접근 레벨에 따라 처리하도록 하고 있다. 이동에이전트의 자원 접근 레벨은 이동에이전트가 시스템 상태를 변경할 수 있는지를 판별하기 위해 사용된다. Admin과 User 자원접근 레벨을 가진 에이전트는 실행 중인 에이전트 시스템의 상태를 변경할 수 있으며, Gypsy 자원접근 레벨을 가진 에이전트는 실행 중인 에이전트 시스템의 상태를 변경할 수 없다.

예외 상황을 받은 에이전트 시스템은 그 에이전트를 생성한 에이전트 시스템에게 예외를 알려준다. 에이전트를 생성한 에이전트 시스템은 자원접근 레벨과 예외상황에 따라 예외 처리를 한다.

〈표 1〉 예외 종류에 따른 에이전트 예외처리 방법

예외 종류	객 체 저장위치	시 스 템 상태변경	처 리 방 법
자원 접근 불 가	에이전트 시스템의 JavaSpace	가 능	에이전트를 생성한 에이전트 시스템에게 통보 후, 사용 중인 자원을 반납하고 에이전트 종료
		불가능	에이전트를 생성한 에이전트 시스템에게 통보 후, 응답 결과에 따라 에이전트 재실행 또는 종료
목적지 에이전트 시스템으로 이동 불가	이동경로관리자의 JavaSpace	가 능	에이전트를 생성한 에이전트 시스템에게 통보 후, 이동경로관리자에게로 이동
		불가능	에이전트를 생성한 에이전트 시스템에게 통보 후, 이동경로관리자에게로 이동
시스템 재시작	에이전트시스템의 JavaSpace	가 능	에이전트를 생성한 에이전트 시스템에게 통보 후, 사용 중인 자원을 반납하고 에이전트 종료
		불가능	에이전트를 생성한 에이전트 시스템에게 통보 후, 응답 결과에 따라 에이전트 재실행 또는 종료

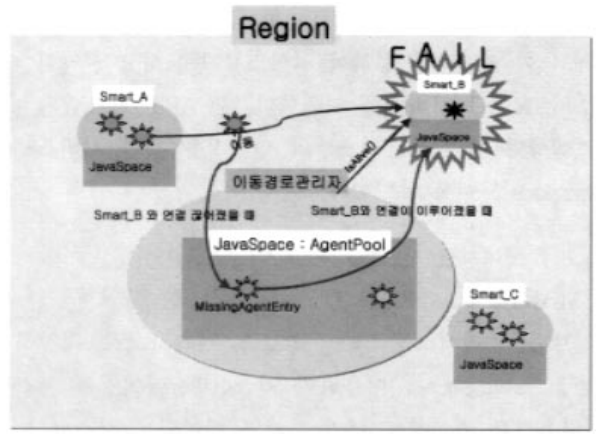
에이전트가 목적지 에이전트 시스템으로 이동을 못하고, 이동경로관리자가 연결되지 않았을 때, SMART 시스템은 에이전트의 예외 상황을 에이전트를 생성한 에이전트 시스템에게 통보 후, 에이전트를 종료한다.

SMART 시스템은 MAF 명세의 MAFAgentSystem 인터페이스에 다음과 같은 메쏘드를 추가하여 이동에이전트에서 발생된 예외를 처리하도록 하였다.

- *AgentExMessage* ***report_Excpetion****(in Name agent_name, in ResourceName, in string message, inMAFAgent System exception_catcher)* : 에이전트는 목적지 에이전트 시스템의 자원 정보와 예외 상황을 에이전트를 , 생성한 에이전트 시스템에게 전달한다.
- *boolean* ***release_resource****(in Name agent_name, in ResourceName resource_name)* : 예외 상황이 발생된 에이전트가 점유한 시스템 자원을 풀어준다.

3.6.3 에이전트 전송 도중 발생 가능한 예외 처리

SMART 시스템은 에이전트가 목적지 에이전트 시스템을 찾지 못할 경우, 각 Region마다 존재하는 JavaSpace (MissingAgent)에 에이전트를 임시적으로 저장한다. MissingAgent 저장소는 이동하려는 목적지 에이전트 시스템과 실행 중이던 에이전트 시스템, 그리고 에이전트를 인자로 하는 Missing AgentEntry를 저장한다. 이동경로관리자 (PathHandler)는 ***isAlive*** (in MAFAgentSystem target_agentsystem)메쏘드을 이용하여 저장된 에이전트의 목적지 에이전트 시스템이 접속가능한지 일정 시간 간격으로 알아보고, 목적지 에이전트 시스템이 접속 가능할 때, 에이전트를 전송한다. 그리고 저장된 MissingAgentEntry의 Lease는 기본적으로 24시간으로 설정되며, 에이전트가 생성될 때 결정된 Lease값으로도 설정된다. 이 시간을 초과하는 MissingAgentEntry는 JavaSpace에서 삭제된다.

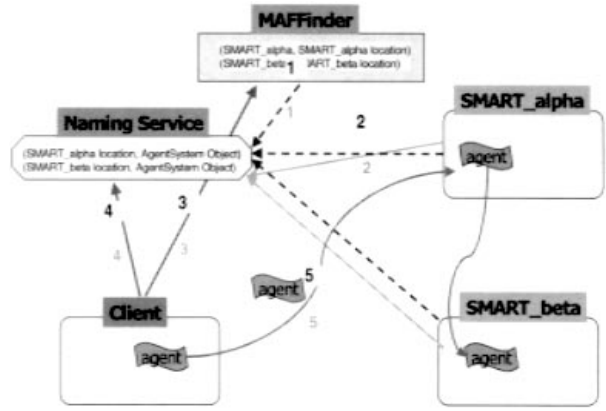


(그림 9) 에이전트 전송 도중 발생 에이전트 처리

4. SMART : 에이전트 프로그래밍

4.1 SMART 시스템의 동작

다음은 SMART 시스템에서 자신의 나이를 출력하는 이동 에이전트를 설명하고 있다. 그 이동 에이전트는 SMART_alpha에서 자신의 작업을 수행한 후 다시 SMART_beta로 이동하여 자신의 작업을 완료하는 간단한 예제 프로그램이다.



(그림 10) SMART 시스템의 동작

1) MAFFinder는 처음 기동할 때, 자신의 객체를 CORBA Naming 서버에 등록한다. 또한 SMART_ alpha, SMART_beta가 처음 시작할 때, 각자는 자신의 객체를 CORBA Naming 서버에 등록한다.
2) SMART_alpha와 SMART_beta는 MAFFinder에 등록시킨다.

1), 2)의 과정을 통하여 MAFFinder와 두 개의 이동

에이전트 시스템이 기동되었다.

3) 클라이언트는 MAFFinder를 통하여 이동에이전트를 이동시킬 SMART_alpha의 주소를 구한다.
4) 목적지 시스템인 SMART_alpha의 receive_agent()를 호출하여 이동 에이전트를 이동시킨다.
5) SMART_alpha에서 자신의 작업을 다 마친 이동에이전트는 SMART_beta의 주소를 구하여 이동한다.

4.2 클라이언트 프로그램

다음은 4.1에서 설명한 SMART 시스템의 동작을 실제 프로그램 코드로서 보여주고 있다. 아래의 코드는 에이전트 생성과 전송에 관련된 클라이언트 측면의 코드이다.

```
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 ..... ①
      orb    = org.omg.CORBA.ORB.init(args, props);
      client = new SmartClient();
  // Create MAFAgentSystem_Impl Object ..... ②
      mafAgentSystem = new MAFAgentSystem_Impl(orb);
  // Get MAFFinder Reference ..... ③
      mafFinder = mafAgentSystem.getMAFFinder();
  // Get location of SMART_alpha agent system ..... ④
      locations = client.lookUp_agentSystem("SMART_alpha");
      dest_mafAgentSystem = mafAgentSystem.getObjectReference(locations[0]);
  // Create Agent!! ..... ⑤
      Name countAge = client.createAgent();
  }
}
```

(그림 11) SMART : Client 프로그램

SmartClient 클래스는 먼저 ORB.init()을 이용하여 서버객체와의 통신을 위하여 ORB를 초기화한다. 이를 통하여 MAFAgentSystem 인터페이스를 구현한 객체인 MAFAgentSystem_Impl은 이동 에이전트를 목적지 서버로 이동시킬 수 있다. lookUp_agentSystem()과 getObject-Reference()를 이용하여 "SMART_alpha"라는 이동 에이전트 시스템의 레퍼런스를 구할 수 있다. createAgent()는 클라이언트에서 countAge라 불리는 하나의 에이전트를 생성한다.

4.3 에이전트 프로그램

다음의 코드는 SMART 시스템에서 에이전트에 해당하는 코드를 보여주고 있다.

```
public class Agent implements Runnable, Serializable {
  public Agent(String name, int age, LinkedList path) {
        agentName    = name;
        agentAge     = age;
        agentPath    = path;
  }
  public void run() {    // Agent Behavior ..... ①
        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 ..②
        }else {   ....  }
  }
}
```

(그림 12) SMART : Agent 프로그램

위의 에이전트 코드는 자신의 나이를 카운트하는 간단한 예제 프로그램이다. 실행 도중에 다른 서버로 이동하더라도 이전에 중단된 상태에서부터 계속 카운트를 실행한다. 에이전트 생성자에서 LinkedList 매개변수는 에이전트가 이동할 경로를 나타낸다. run()은 에이전트가 수행해야할 행동을 나타내고, gotoNextHost()를 이용하여 다음 서버로 이동할 서버의 이름을 찾는다.

4.4 서버 프로그램

SMART 시스템의 서버 측 코드는 다음과 같다.

```
public class Smart {
  public static void main(String[] args) {
            :
    // ORB and BOA Initialization          ①
    orb = org.omg.CORBA.ORB.init(args, props);
    boa = ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);

    // Create MAFAgentSystem_Impl Object          ②
    mafAgentSystem = new MAFAgentSystem_Impl("Smart_alpha");
    boa.impl_is_ready(null);
  }
}
```

(그림 13) SMART : Server 프로그램

Smart 클래스는 "Smart_alpha"라는 서버 프로그램을 가동시킨다. 다른 이동 에이전트시스템 또는 클라이언트와 통신을 하기 위하여 ORB.init()과 BOA_init()을 호출하여 초기화한다. 그리고 boa.impl_is_ready()는 서비스 요청을 기다린다.

4.5 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 ..... ①
        while(true) {
            Agent agent = getAgent();
            Thread agentThread = new Thread(agent);
            agentThread.start();           // Execute Agent ..... ②
        }
    }
    // Put a Agent in the agentPool ..... ③
    public synchronized void putAgent(Agent anAgent) {
        agentPool.add(anAgent);
        notify();
    }
    public synchronized Agent getAgent() {  // Get a Agent ... ④
        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 14) SMART : Place 프로그램

*Place 클래스*는 에이전트를 agentPool에 저장시키기 위한 putAgent()와 저장된 에이전트를 하나씩 가져오는 getAgent() 두개의 메쏘드를 제공하고 있다. 플레이스는 이 두개의 메쏘드를 이용하여 에이전트의 저장과 실행의 동기화를 효율적으로 지원하게 하였다. run()은 에이전트를 실행시키기 위한 쓰레드를 하나 생성하여 에이전트를 실행시킨다.

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 지금 까지 개발된 시스템들의 특징을 설명하고, 각 시스템에서 사용한 이동 에이전트의 이동성 기법에 대하여 비교한다.

IBM에서 개발한 에이전트 시스템으로는 *Aglets*이 있다[15]. 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-Go*이다[10]. 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 위한 Java 기반의 시스템이다[16]. 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[11]. 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[17]. Grasshopper 시스템은 RMI, Socket, IIOP 등 다양한 통신 채널을 제공한다.

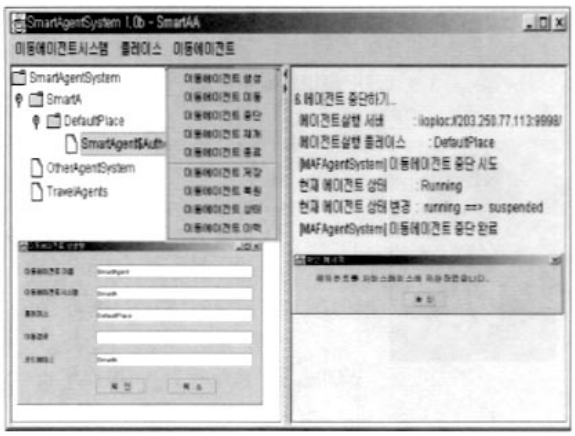
〈표 2〉 에이전트 시스템 비교

에이전트 시스템	상호 운용성	코드 이동성	보 안 정 책
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, ResourceManager)

6. SMART 시스템 실행 모니터링

SMART 시스템의 기동된 전체 모습과 기능을 (그림 14)와 (그림 15)에서 자세히 보여주고 있다.

SMART 시스템은 자신의 IP주소, 포트번호, 시스템 이름, 그리고 CORBA 네이밍 주소 정보를 가지고 기동된다. 전체 프레임의 구성은 에이전트 시스템, 플레이스 그리고 에이전트에 해당하는 노드를 표현한 트리 패널과 각각의 노드 정보를 나타내는 정보 패널, 그리고 메뉴바로 구성된다.



(그림 15) 이동에이전트 생성과 이동

(그림 14)는 이동에이전트 생성 메뉴를 통하여 에이전트 생성 기능과 에이전트 상태 정보, 에이전트에 대한 제어 정보를 정보 패널에 보여주고 있다.

(그림 15)는 이동 에이전트를 목적지 이동 에이전트 시스템으로 이동시키고, DefaultPlace 노드에서 TravelAgents노드로 이동됨을 보여 주고, 이 노드의 정보를 통하여 현재의 에이전트의 위치와 상태 정보를 알아 볼 수 있는 기능을 제공하고 있다. 그리고 이동시킨 이동 에이전트가 목적지 에이전트 시스템의 자원을 접근할 수 없는 상황을 알려주고 이를 처리하는 모습을 보여준다.

SMART 시스템에서 에이전트의 이름, 현 위치, 현 상태, 그리고 이동한 경로 이력 등의 에이전트에 대한 정보 보기 기능을 제공하고 있다.

(그림 16) 이동에이전트 정보 및 예외 처리

7. 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 시스템들 간의 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 시스템을 개발하였다. MAF 명세는 MAFAgentSystem과 MAFFinder 두 개의 인터페이스로 이루어져 있으며, 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, 자원관리자, 보안관리자 그리고, 모니터의 네 개의 모듈로 이루어져 있다. 플레이스는 쓰레드 그룹을 이용하여 에이전트의 수행을 효율적으로 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. 보안관리자는 OMG MAF에서 요구하는 4가지 정책을 Protekt 3.0 SSL 패키지와 SMART 시스템의 로그파일을 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.

SMART 시스템의 견고성을 높이기 위하여 에이전트의 예외처리와 영속성(persistence) 기능을 제시하였다. 에이전트 시스템에서 일어날 수 있는 예외상황은 실행 중인 에이전트가 자신의 자원접근등급에 어긋난 자원을 액세스하고자할 때 발생할 수 있는 예외, 이동하려는 목적지 에이전트 시스템을 찾지 못할 때 발생하는 예외, 그리고 에이전트 시스템의 비정상 종료 시 발생하는 예외와 에이전트 시스템의 영속성을 JavaSpace를 이용하여 효율적으로 처리할 수 있는 기법을 제안하고 구현하였다.

SMART 시스템은 현재 에이전트 또는 에이전트 시스템들 간의 일대일 통신만을 지원하고 있다. 그리고 에이전트의 영속성을 위하여 JavaSpace를 이용하고 있다. 앞으로 이러한 기법을 더 발전시켜 서버의 실패나 네트웍의 분할

(partition)이 발생하여도 안정된 서비스를 제공할 수 있는 통신 채널을 이용하여 에이전트간의 통신, 에이전트 시스템 간의 통신을 지원할 예정이다[18].

참 고 문 헌

[1] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, "Understanding Code Mobility," IEEE Transaction On S/W Engineering, Vol.24, NO.5, May 1998.

[2] Mobile Agent System Interoperability Facilities Specification, OMG Inc, 1998. 3.

[3] Antonella Di Stefano, Lucia Lo Bello, Corrado Santoro, "Naming and Locating Mobile Agents in an Internet Environment," IEEE 0-7803-5784-1/99, 1999.

[4] B.Venners, "The Architecture of Aglets" JavaWorld, http : //www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html, April 1997.

[5] T. Finin, Y. Labrou, Y. Peng, "Mobile Agents Can Benefit from Standards Efforts on Interagent Communication," IEEE Communications Magazine, July 1998.

[6] Neeran M. Karnik, "ecurity in Mobile Agent Systems," PhDS thesis, University of Minnesota, October 1998.

[7] Netscape Inc, "Introduction to SSL," "http : //developer.netscape.com/docs/manuals/security/sslin/index.htm," 10. 1998.

[8] Neeran M. Karnik, Anand R. Tripathi, "Design Issues in Mobile Agent Programming Systems," University of Minnesota Minneapolis, June 1998.

[9] Krishna Sankar, "Java 1.2 Class Libraries Unleashed Vol I, II," Sams, 1999.

[10] William Li, David G. Messerschmitt, Java-To-Go Mobile Agent System, University of California at Berkeley, 1998

[11] S. Choy, T.Magedanz, "Grasshopper Technical Overview," IKV++ GmbH, February 1999.

[12] Forge Information Technology, "Protekt Encryption 3.0 Programming Guide," 1999.

[13] Danny Ayers, Hans Bergsten, "Professional Java Server Programming," Wrox Press Ltd, 1999.

[14] Edwards, W. Keith, "Core JINI," The Sun Microsystems press Java Series, 1999.

[15] Gunter Karjoth, Danny B. Lange, Mitsuru Oshima, "Security Model For Aglets," IEEE Internet Computing, 1997.7.

[16] Anand R. Tripathi, Neeran M. Karnik, Manish K. Vora, Tanvir Ahmed, and Ram D. Singh, "Ajanta - A Mobile Agent Programming System," 1998.

[17] C. Baumer, M. Breugst, S. Choy, T. Magedanz, "Release 1.2 Basics and Concepts," Grasshopper, February 1999.

[18] 안건태, 문남두, 정현락, 유양우, 이명준, "JACE 그룹통신시스템을 이용한 신뢰성 있는 공유객체공간의 개발", 한국정보과학회 99가을 학술발표논문집(III), pp.218-220, 1999.

유 양 우

e-mail : soft@mail.ulsan-c.ac.kr

1995년 경일대학교 전자계산학과 졸업 (공학사)

1997년 울산대학교 전자계산학과 졸업 (공학석사)

2000년 울산대학교 전자계산학과 박사과정 수료

2000년~현재 울산과학대학 컴퓨터정보학부 근무(전임강사)

관심분야 : 이동 에이전트 시스템, 그룹통신 시스템, 분산객체 프로그래밍 시스템 등.

김 진 홍

e-mail : karif99@dreamwiz.com

1999년 울산대학교 전자계산학과 졸업 (학사)

2001년 울산대학교 컴퓨터정보통신 공학부 졸업(석사)

2001년~현재 울산대학교 컴퓨터정보통신 공학부 박사과정

관심분야 : 이동 에이전트 시스템, 웹 프로그래밍, 분산객체 시스템, 생물정보학 등.

구 형 서

e-mail : masker@hanmail.net

2001년 울산대학교 컴퓨터정보통신공학부 졸업(학사)

2001년~현재 울산대학교 컴퓨터정보통신 공학부 석사과정

관심분야 : 이동에이전트 시스템, 웹 프로그래밍, 생물정보학 등.

박양수

e-mail : yspk@uou.ulsan.ac.kr

1978년 울산대학교 전자계산학과 졸업 (학사)

1981년 서울대학교 계산통계학과 졸업 (석사)

1986년~현재 서울대학교 계산통계학과 박사과정

1980년~현재 울산대학교 컴퓨터정보통신 공학부 근무 (부교수)

관심분야 : 분산처리, 컴퓨터알고리즘 등.

이 명 재

e-mail : ymj@uou.ulsan.ac.kr

1987년 서울대학교 전산학과 졸업 (학사)

1989년 서울대학교 전산학과 졸업 (석사)

1995년 서울대학교 전산학과 졸업 (박사)

1996년~현재 울산대학교 컴퓨터정보통신 공학부 근무 (부교수)

관심분야 : CBSE, WBSE, 분산객체 시스템

이 명 준

e-mail : mjlee@uou.ulsan.ac.kr

1980년 서울대학교 수학과 졸업(학사)

1982년 한국과학기술원 전산학과 졸업 (석사)

1991년 한국과학기술원 전산학과 졸업 (박사)

1982년~현재 울산대학교 컴퓨터정보통신 공학부(교수)

1993년~1994년 미국 버지니아대학 교환교수

관심분야 : 프로그래밍언어, 분산 객체 프로그래밍 시스템, 병행 실시간 컴퓨팅, 인터넷 프로그래밍 시스템 등.